NASA CR-165836

Software Reliability: Repetitive Run Experimentation and Modeling

Phyllis M. Nagel
James A. Skrivan

Boeing Computer Services Company
Space and Military Applications Division
Seattle, Washington 98124

Contract No NASI-16481
February 1982

National Aeronautics and
Space Administration

Langley Research Center
Hampton Virginia 23665

All Blank Pages

Intentionally Left Blank

To Keep Document Continuity

BCS-40366

SOFTWARE RELIABILITY: REPETITIVE RUN EXPERIMENTATION AND MODELLING

Prepared Under Contract NAS1-16481

By

Phyllis M. Nagel
James A. Skrivan

Boeing Computer Services Company
Space and Military Applications Division
Seattle, Washington 98124

For

National Aeronautics and Space Administration

February 1982

PREFACE

The authors are much indebted to many of their colleagues in the Consulting Division and the Space and Military Applications Division of BCS for their assistance and counsel. In particular, special gratitude should be extended to:

- Mr. John R. Brown for his management of the project and his long-standing support and understanding of the problem.
- The two programmers for denying many of their professional instincts for timely debugging in order to support the goals of this experiment.
- Mr. Kerry Whitaker for his contributions in sizing and designing the original experiment, and
- Dr. Roberto Altschul for encouraging interest in log linear models and for the methodology for estimating the nonlinear parameter set.

Finally the authors would like to thank Dr. D. R. Miller of the Department of Operations Research, George Washington University, for his many stimulating and exceedingly helpful discussions.

CONTENTS

CONTENTS (Continued)

CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

1.0 SUMMARY AND INTRODUCTION

1.1 SUMMARY

Boeing Computer Services has conducted carefully designed and controlled software-development experiments with analysis in support of software-reliability estimation and modelling. Two programmers individually designed and coded three FORTRAN programs each from three problem specifications. These programs were then executed in repetitive run sampling, where a run is a sequence of interfailure times recorded on each of a series of program states.

The run data has been used to verify that interfailure times are exponentially distributed, to obtain good estimates of the failure rates of individual errors and to demonstrate how widely the rates vary. This latter fact invalidates many of the popular software reliability models now in use. In addition, it was observed that the log failure rate of interfailure time was nearly linear as a function of the number of errors corrected.

Cox's proportional hazards model has all of the observed characteristics of the experiment data and is proposed as a new model of reliability. Maximum likelihood estimates for the unknown parameters were obtained for all six programs using nonlinear optimization techniques. Statistical tests on these estimates indicate there are strong programmer and problem effects on the background failure rate. Results also show that over 80% of the observed variation in the logarithm of the failure-rate data can be explained by the proportional hazards model. A tentative physical predictor was proposed based on Halstead's information criteria N which might be used in forecasting model parameters.

1.2 INTRODUCTION

Since the importance of good models for predicting software reliability is undeniable, it is an anomaly that so little model development has been based on insights gained from specific experimental results. Predicated on the belief that there is much to be learned regarding the man-in-the-loop process of software development before modeling can be effective, an experiment has been conducted that both attempts to extend the information base and provide experimental verification of some of the popular assumptions regarding the failure structure of software. This experiment differs from other software experiments in several regards. The emphasis is on the prediction of software reliability from the traditional point of view of risk assessment rather than as a management tool in software development. The experiment utilizes the concepts of formal statistical design and experimental control and by replicating the design makes it possible to examine the detection process when errors are not identically distributed.

The design of this experiment has been chosen to explore only a few of the issues of software reliability and does not try to include the entire set of factors thought to influence the software error structure. Two primary self-imposed constraints were adopted. First of all, by concentrating on reliability assessment, the experiment consciously avoids all of the issues behind time and accuracy growth of software undergoing redefinition, and explores only those related to problems with fixed specifications. Secondly, the experiment assumes that the detection of errors in a

program depends only on the interaction of the error set, the criteria or detector establishing correctness and the probability of detection of each error given the detection mechanism. Thus, except for the process of error removal, the probability of detection for a specific error remains fixed throughout the experiment.

Data from software reliability tests has traditionally consisted of a record of the successive interfailure times following the detection and correction of sequential errors. In the present context such a set of data will be called a program run. Since fixing a program by correcting an error is equivalent to a program design change, a run of data consists of a single observation on life length for each of a sequence of design configurations that are different but possibly connected. It is this fact that has inhibited the statistical investigation of the properties of software reliability.

The experiment reported here enriches the data base by increasing the sample size of each of the design stages. This has been accomplished by replicating runs. Each run consists of a reinitialization of the program to its original design configuration and a repetition of the process of obtaining interfailure times with different independent sequences of randomly generated input data.

With this data several modeling properties of the software failure phenomenon have been investigated. In particular these include investigations of:

a. The exponential assumption as the distribution of interfailure time.

b. The assumption of equal error probability implicitly assumed in the Jelinski-Moranda [1], Shooman [2], and Schick-Wolverton [3] models versus the unequal error detection probability proposed by Littlewood and Verrall [4] and further investigated by Littlewood [5].

c. The consequences of the results of a. and b. (above) on reliability considerations.

d. The effect of programmer on the error characteristics for a fixed problem specification.

e. The effect of problem on a given programmer's error performance.

f. The number of program executions to failure versus interfailure time measured in computer resource units (CRU's) as a base in which to measure reliability, where here, a program execution means a complete processing of the input case from the beginning of the program to the final output, correct or incorrect.

g. Potential physical features of the program to use as predictors of the error structure.

This report is presented in six sections. Section 2 contains a detailed description of the experiment with reference to its design and the controls imposed to insure data integrity. Sections 3 and 4 provide details of the actual experiment. Section 3 describes the computer programs written to maintain the flow of the experiment and conduct error detection. Section 4 outlines the problems and specifications selected for this experiment together with the experience histories of the programmers

involved and their programs written from the specifications. This section also includes some discussion on the nature of the errors manifested during the experiment. In Section 5, descriptive statistics of the observed data are provided and the investigations referenced above are described. Details are then given of a proposed new model of software reliability. Section 6.0 provides conclusions of the study.

2.0 EXPERIMENT

2.1 BASIC HYPOTHESIS ON ERROR DETECTION

By assuming that failure rates are simple functions of the total number of errors in a program, currently popular software-error models imply implicitly that errors are equally probable. The belief in this property has not only influenced the modeling of software reliability, it has also influenced the design of software-error experiments and the types of data that have been collected to date. Based on the complicated structural relationship that exists between an error and the forces causing its manifestation, it seems quite possible, however, that errors are not equally probable. This experiment therefore has been designed specifically to explore the exact nature of these probabilities.

2.1.1 Relationship Between the Error Set and the Input Set

There is a correspondence between an error in the program and a subset of the set of all possible inputs to the program whose elements can detect the error where the word input stands for the complete vector of quantities required for a single program execution. Some subsets no doubt are larger than others and some may only consist of a single input. Sets can overlap if some inputs can detect more than one error and therefore the correspondence between the subsets and the errors is not necessarily one to one. One subset contained in another may mean that the contained error is nested and cannot be detected until the other error is first corrected.

Depending on the experiment, it may or may not be the case that the incidence of an input in the intersection set of two or more errors causes the correction of all of the errors involved. In this experiment, however, all errors detectable by a given input are corrected before testing is resumed.

If an error exists that can only be detected by a region of the set of inputs to the program that is rarely used in practice, the probability of detection is necessarily small. It is also true (for experiments designed as this one is) that the unreliability of a program containing a fixed error set is the probability of the union of these error-detecting subsets of the input set.

2.1.2 Usage Distribution

Once a program has been declared operational the dynamic influences on the development of the program are stabilized. At this time the program is intended to be used in an operating environment defined in terms of the values and frequencies of the inputs selected for execution. It is relative to this operating environment, then, that the question of reliability is posed, and if the environment changes so will the statements regarding reliability.

The set of all potential uses of a program imposes a probability distribution on the set of all possible program inputs. This reflects that some regions of the set of inputs are more interesting or more popular than others. This distribution, sometimes called the "usage" distribution, is described in Brown and Lipow [6]. Since there is a correspondence between an error in the program and a subset in the input set whose elements can detect the error, the probability of detection of an error is the

probability of sampling at random, according to the usage distribution, in its associated subset.

The usage distribution, therefore, plays a critical role in determining error probabilities and it is quite possible that some errors are more probable than others either because their associated input subsets are larger or because their subsets are more likely to be selected in execution. Furthermore, all statements regarding the reliability of a program are really referencing the interplay between the probabilistic forces of the usage distribution and the error structure of the program.

2.1.3 Error Detection

Since it is true that the detection of an error implies the existence of a detector, then it also follows that the type of detector in use influences the probability of detection and therefore has an impact on the reliability of the program. That is, in the sense of the user's knowledge of an error as opposed to its fundamental existence, the detector defines the error.

2.1.4 Sampling Method

Three major influences on the reliability of a program have been discussed: the existing error set, the usage distribution and the error-defining detector that judges output correctness. There is, however, one other factor that can be extremely important in some contexts that has to do with the nature of the sampling from the usage distribution. In most experiments described in the literature, sampling proceeds by independent sampling with replacement and reliability is unaffected. In some fields of application, however, such as avionics, there is time dependence in the input stream. The existence of this autocorrelation in the sampling can exercise considerable influence over a program's reliability performance. A manifestation of this potential influence is the concept of software-error "bursts"[7].

2.2 EXPERIMENTAL FEATURES

2.2.1 Overall Controls

The specifications for each of the problems considered in the experiment included complete definitions of the usage distributions to be employed and all detectors. Controls on the error set included controlling for problem, controlling for programmer and controlling the definition of a program's initial state in terms of the test cases it must successfully pass. Data was sampled independently and with replacement at all times.

2.2.2 Run Repetitions

Traditional software experiments remove errors without allowing for the probabilistic impact of the detected error on subsequent error detection. In order to be able to measure this impact the present experiment was designed to provide some information about "what was there" from a probabilistic point of view. This was achieved by repetitively sampling a program from its initial error state (state 0) through n errors, obtaining a series of runs. A run then consists of n observed times to failure on each of n sequential program stages.

2.2.3 Detection Order Effects

This method of sampling introduces two other concepts which must be defined, namely those of program stage and program state. A program stage refers to the number of errors that have been corrected since sampling began with the program in its initial state. A program state is a listing of errors by number that have been detected and corrected since the initialization. In traditional experiments the concept of state was not recognized. In this experiment observing runs makes it possible to not only study how errors are distributed but also to study the consequences of removing errors in a random order.

Time to next failure of a program is conditional on the error state of the program at the time of the last failure. Since the order of error removal is random, it is of interest not only to study the magnitude of the error probabilities but also the effect of order on these estimates. For this experiment not only is the interfailure time recorded at the time of program failure, but also each observed error is diagnosed and recorded by number so that the exact conditioning is known at all times.

2.2.4 Initialization and Static Detectors

Once the program is released from the programmer, the code must satisfy all compiler checks and correctly execute a set of predetermined "reasonable" test cases, numbering at most three. These tests form a set of static detectors that are intended to stabilize the initial error set of each program in order to form a common initial base from which to compare different programs as a function of the number of errors corrected. In traditional experiments there has been little attempt to control for a common 'initial denominator', and comparing error structures across different programs was further thwarted in this regard.

Later in the paper a second initialization criterion is proposed that seems even more useful at providing a stronger, more consistent basis for comparing dissimilar programs.

Other static detectors such as the program DAVE [8] were considered as joint detectors with the above set to insure greater commonality in the initial state. However, the potential return compared to the cost of their use did not seem justified until the problem is better understood.

2.2.5 Dynamic Detectors

For each of the problems selected for this experiment, an existing program satisfying the specifications for the problem has been in use for some time. The output of this pre-existing program, executed with identical inputs to the program on test, is used as a comparator to determine the correctness of the new program. Although absolute correctness in any sense cannot be assured by this method, a program that has received extensive prior use has already been through many and varied tests in its lifetime and is felt to be a reasonable norm. The use of other norms, however, can change the judgment of a program's reliability.

In addition to this comparator, the operating system itself has some detecting ability that causes the program to stop, or "bomb off", during execution. The familiar system-interrupt message, "division by zero," is an example of a failure detected by

an operating system. This detecting ability must be considered as part of the dynamic detecting environment that defines errors.

Other dynamic detectors were considered, particularly those employing the technique of dynamic assertion to test selected program variable states [9]. Again the use of these detectors seemed unjustified at this time on the grounds of cost, although the technique seems useful and may be considered at some future date.

2.2.6 Debugging and Static Debug Tests

When an incorrect execution of the program is detected, the inputs to the program are saved while the bug is corrected. The corrected program is then subjected to the same static tests as in the original set. That is, it must satisfy all compiler checks and correctly execute the original test cases. In addition, it must also correctly execute the saved last input case. An incorrect execution at this time indicates one of two possibilities: the bug causing failure has not yet been corrected; or the saved input is causing an additional simultaneous error from one or more errors in the original error set.

By saving the last input case and forcing the correction of all errors detected by this input, the experiment can both provide a measure of the probability of the intersection and give a slightly better estimate of the marginal error probabilities.

To clarify this last observation it should be noted that in many situations one error can dominate another due to the ordered nature of a program's execution. That is, one error is always detected before another when an input is selected in the intersection. By referencing a Venn diagram involving the intersecting sets A and B, if A dominates B and the inputs are not fully explored to determine when they fall in the intersection, the program measures the probability of B as $P(\bar{A} \cap B)$ before A is observed and $P(\bar{A} \cap B) + P(A \cap B)$ after A is removed. On the other hand, by exploring if a given input is in the intersection, the probability of B is measured the same before or after the dominating error is removed.

2.2.7 Ripple Effect

When an execution of the program is declared in error, the nature of the error is investigated to determine if the error was introduced at an earlier stage as part of the debugging process. If so, separate books are kept of the error in order to track the growth of unreliability as well as the growth of reliability if this seems indicated.

2.2.8 Flow of Experiment

Figure 2.2.8-1 illustrates the flow of the experiment for a single problem and a single programmer's code. The process is in fact based on a simulation until failure of various program states. It begins with an initial version of the program that has successfully passed all of the static tests. Random input is then generated according to the usage distribution defined in the problem specifications and the program is executed with these inputs. Simultaneously, the comparator is executed with these same inputs and correctness of the program on test is determined. If correct the process is repeated with new inputs generated independently until an error is detected. Once detected the error is analyzed, recorded in the proper account with an error number and corrected. At the point where all static tests including the

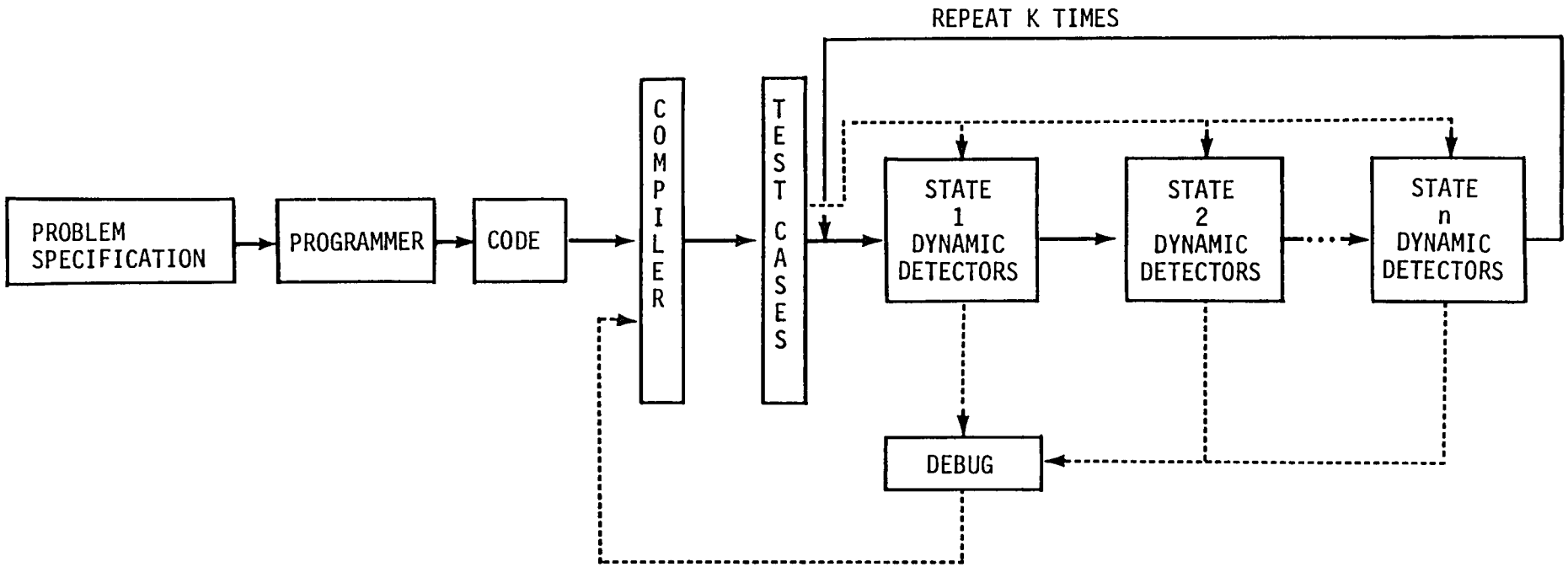


Figure 2.2.8-1. Experiment Flow Diagram.

re-execution of the last input case have correctly executed, simulation can begin on another cycle with the program in this new state.

It was the intent originally to continue cycling until 5 or 6 bugs had been discovered or truncate the process at the predetermined number of executions. In practice neither of these rules was followed too closely, and termination usually occurred when a bug was discovered that was extremely time consuming to correct. Most programs manifested 5 or 6 errors before this occurred.

Repeated runs on the same program are conducted starting from the same initial state, i.e., state 0. The experimental flow for each run is exactly the same but the simulated input stream is generated independently and hence different for each run. Thus each run has the opportunity of generating different random errors in different orders. As a consequence each run represents a random walk through the set of possible errors, and therefore experiences random conditioning at each stage.

2.2.9 Run Sizing

In order to determine how many runs were sufficient to obtain stable estimates of the error probabilities, it was decided to base the size on the probability structure that exists initially since all runs are in a common state during this stage. Assuming that each execution of the program has a fixed probability p of failure, then the probability that the first failure will occur on the i'th execution is expressable in terms of the geometric distribution:

$$P(i) = (1-p)^{i-1}p$$

For k independent runs, let $i_1, i_2, \ldots, i_k$ be the k values each expressing the number of times the program was executed in its initial state before failure occurred. Then the maximum likelihood estimate for p is

$$\hat{p} = \frac{k}{\Sigma i_j}$$

It can be shown that the asymptotic variance of $\hat{p}$ is

$$\text{Var}(\hat{p}) = \frac{p^2(1-p)}{k}$$

Therefore an asymptotic 95% confidence interval on the maximum likelihood estimator is

$$\hat{p} \pm 1.96\sqrt{\frac{p^2(1-p)}{k}}$$

If k is chosen on the basis of the half length L of the resulting confidence interval, then since

$$L = 1.96\sqrt{\frac{p^2(1-p)}{k}}$$

k can be calculated from

$$k = \frac{3.84p^2(1-p)}{L^2}$$

If $p < .05$ and $k = 50$, then $L < .014$. Similarly if $p < .01$ and $k = 50$, then $L < .0028$. Thus $k = 50$ was selected as sufficiently discerning.

2.2.10 Problem and Programmer as Design Factors

To control for the effect that programmer and problem may have on the corresponding error structure of a program, a factorial design consisting of two programmers A and B, each programming the same three problems, labelled 1, 2 and 3, from their specifications, formed the basic experiment. Each of the six programs A1 through B3 thus formed were executed through 50 runs each of which consisted of at least four stages. Within time constraints, the programmers designed, coded and tested their programs using FORTRAN IV.

2.2.11 Data Base

The data base developed from this experiment consisted of a record, for each of the six programmer-problem combinations, of the number of executions until failure for each stage, the interfailure time (in terms of both the number of input cases to failure and computer-resource units (CRU's)), and the number of the error causing failure. A number was assigned each new error as it was encountered. This data base then provided complete knowledge of the error state of the program at each stage and detailed knowledge of the time to failure of the program conditioned on each of these states.

3.0 OPERATIONAL DESCRIPTION

3.1 GENERAL

In order to gather statistics on software failure detection/error correction, the subject programs must be embedded in a software-test environment (Figure 3.1-1). There is a unique, but quite similar environment for each of the three problem specifications. In reality, the subject programs are subprograms which are called in three separate programs: 1) test driver; 2) experiment driver; and 3) error identifier. The latter two programs are indirectly linked via the failed-cases file -- the experiment driver writes all failed cases to the file and the error identifier reads those failed cases. These three overhead programs and the failed-cases file are discussed in the following sections.

The experiment was run on two computer systems: 1) the Boeing Intelligent Terminal System (BITS); and 2) the DEC VAX/VMS System. BITS, a microcomputer-based system, was chosen for the first problem because of fixed rental and sole use by the study. VAX/VMS, a virtual-memory minicomputer-based system, was used because of the need for more storage and faster execution time. These systems and the support software are described in Section 3.3.

3.2 OVERHEAD PROGRAMS

3.2.1 Test Driver

The test driver is the program which determines if the subject programs pass the predefined test cases to bring the subject program to state 0. The principal components of the test driver are: 1) test-case data; 2) correct version; 3) subject program; and 4) comparator to determine correctness. The macro flowchart is shown in Figure 3.2.1-1.

3.2.2 Experiment Driver

The experiment driver is the program which executes the subject program using randomly-generated data, and determines correctness by executing the correct version and comparing the results. The principal components of the experiment driver are: 1) parameter input; 2) input-data generator; 3) correct version; 4) subject program; 5) comparator and 6) results writer. The macro flowchart of the experiment driver is given in Figure 3.2.2-1.

3.2.2.1 Input-Data Generator

Each of the three problem specifications contains descriptions of the joint distribution of the input-data items. Random data from these distributions are obtained using a standard library random number generator.

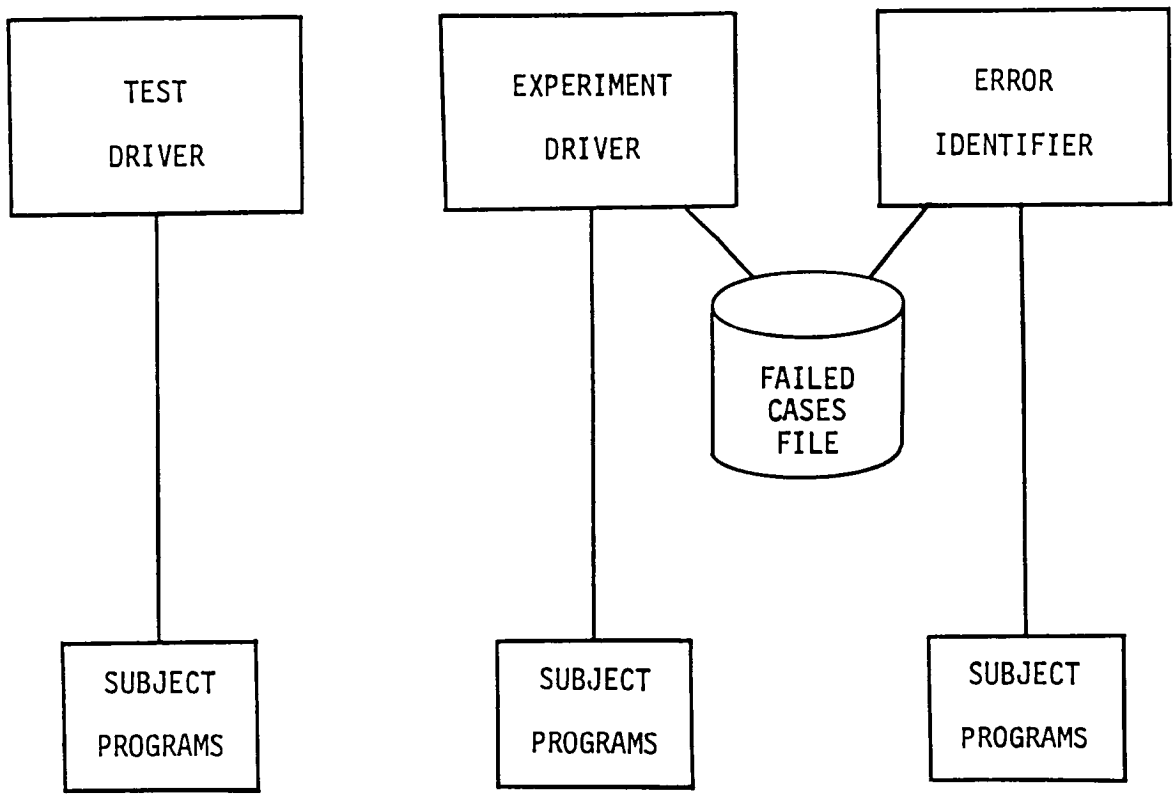


Figure 3.1-1. Experiment Software Structure.

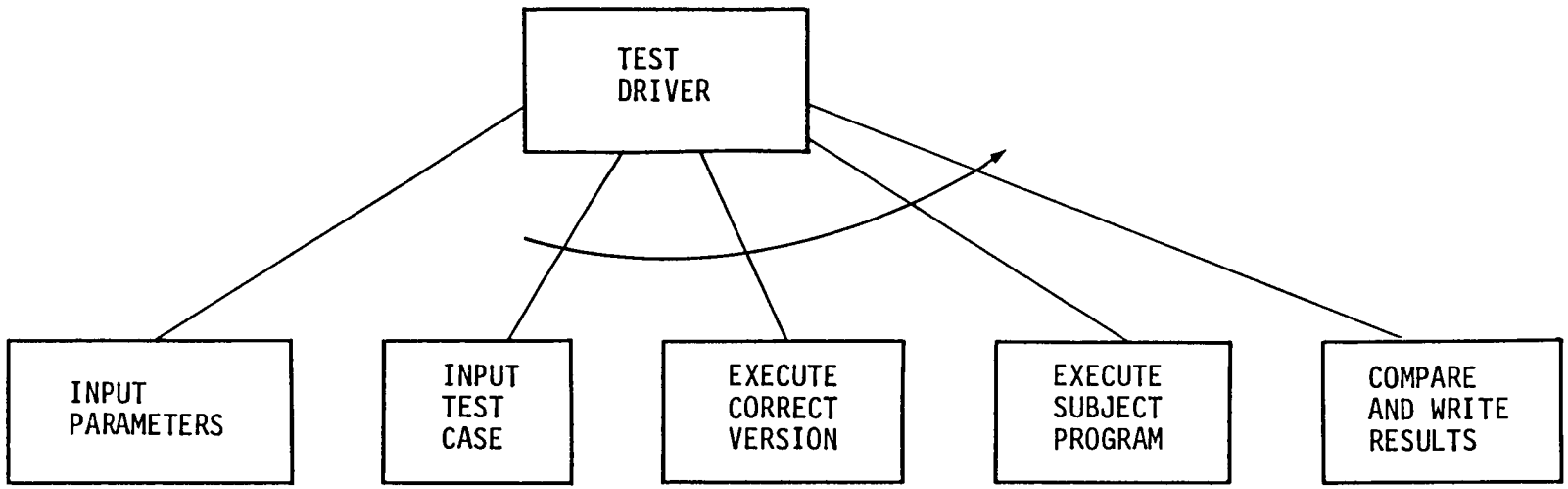


Figure 3.2.1-1. Test Driver Macro Flowchart.

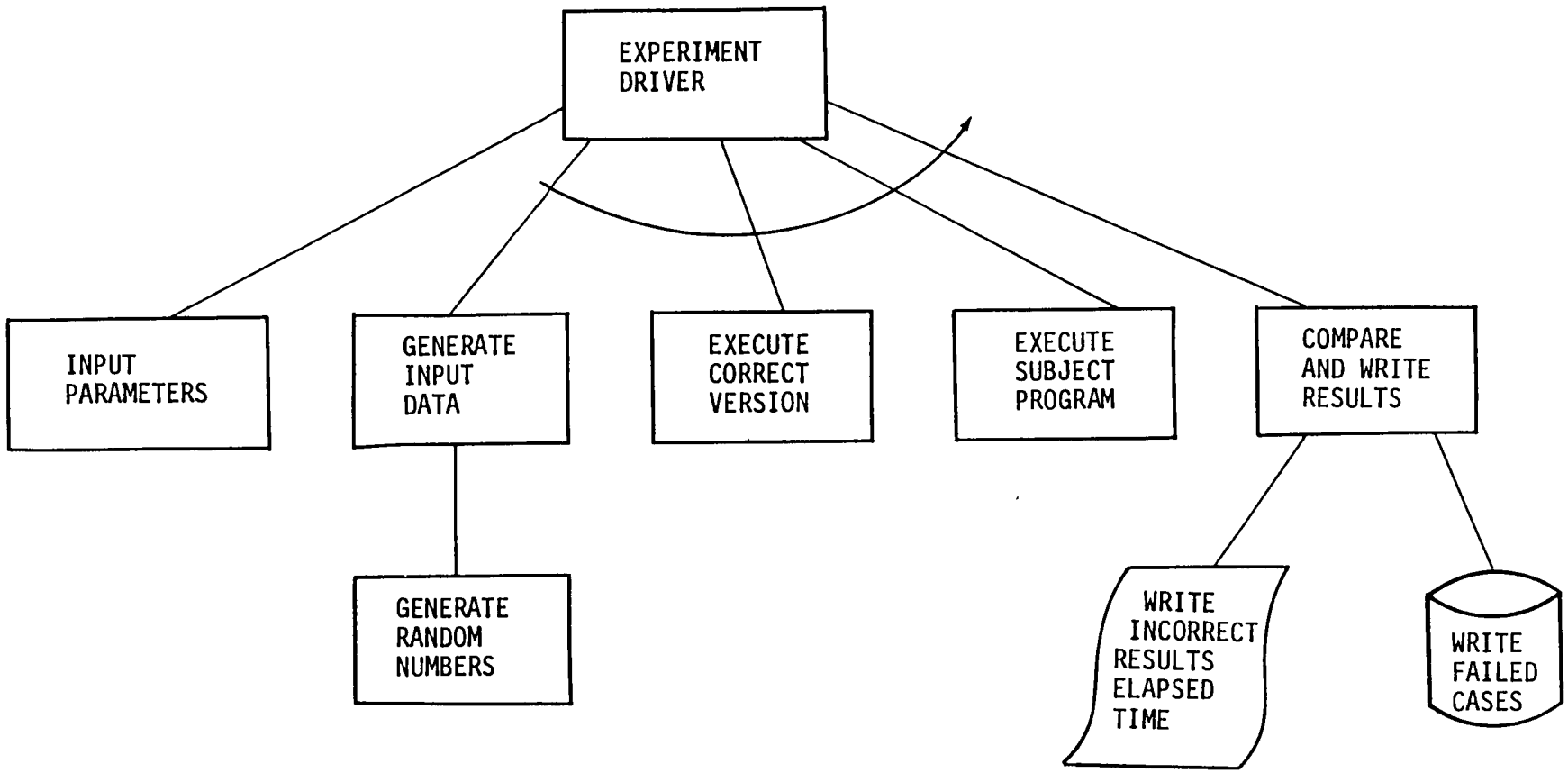


Figure 3.2.2-1. Experiment Driver Macro Flowchart.

3.2.2.2 Comparator

The comparator in the experiment driver is used to determine correctness of the subject program. Simply stated, the comparator is program code to compare every output-data item of the subject program with the corresponding item of the correct version. An error flag is set when a difference in output is noted.

3.2.2.3 Results Writer

The results writer is code in the experiment driver to record the following information: 1) incorrect output as the result of a software failure; 2) number of input-data cases run to software failure; 3) CRU's to the software failure; and 4) failed input-data case.

The incorrect output helps to identify the potential software error(s) causing the failure. The identification of errors and their associated error numbers in each run required the careful matching of various program states with the failed input-data cases. By running selected program states with the failed cases and recording successes, the error numbers can be accurately assigned. Multiple errors can also be identified with this procedure, or equivalently, computer program. The program, called the error identifier, is described in 3.2.3.

The recorded failed case is actually the initial seed of the random number generator associated with the failed case. The volume of written data is thus greatly reduced. However, the price paid is the execution time required in the error identifier to regenerate the full input data of the failed case.

These data are written on the failed-cases file which is permanent disk storage. This file is then easily referenced by the error identifier program at a later time.

The failed-case file is organized by the subject program states that produced the failures. In other words, all the failed cases from a particular program state are grouped together on the file, with some header information identifying that group of cases.

3.2.3 Error Identifier

The error identifier is the program which executes any state of the subject program using the failed-cases file. The structure of the error identifier is much like the experiment driver, in that they both use the correct version and a comparator to determine success. The macro flowchart is given in Figure 3.2.3-1.

The identification of the software errors corresponding to all failures of a particular subject program state involves multiple executions of the error identifier using the program in other states. The following steps constitute the runs of the experiment to identify errors.

1. Run the experiment driver with a subject program in a given state, producing a failed-cases file F.

2. Correct one software error E in the program in state S, which yields the program in state S.E.

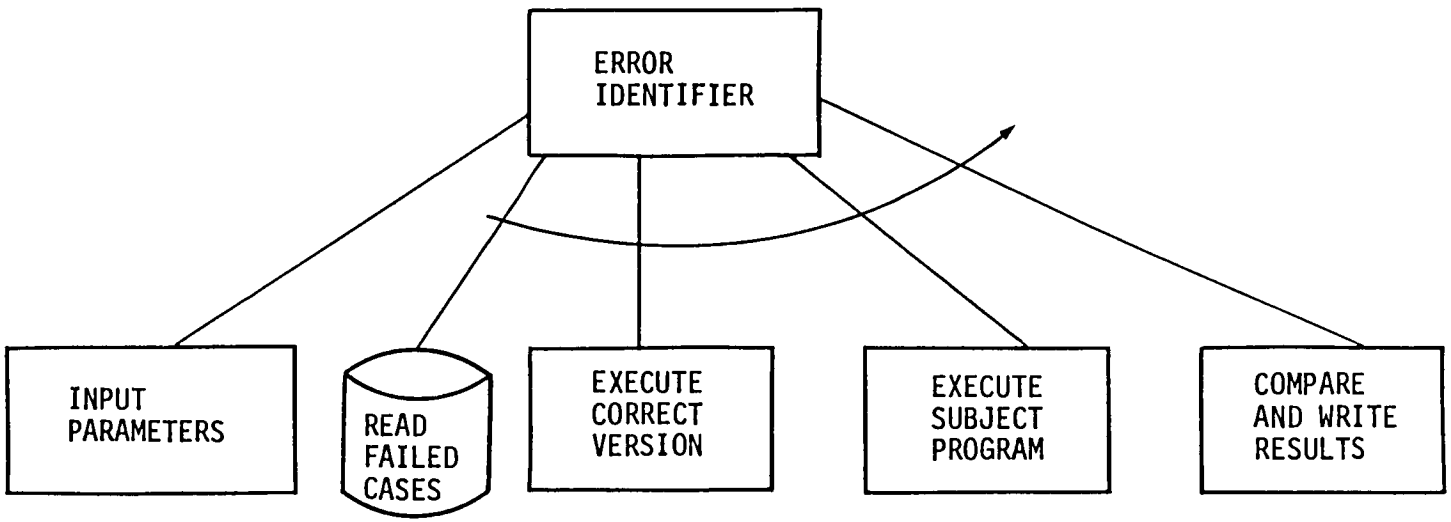


Figure 3.2.3-1. Error Identifier Macro Flowchart.

3. Run the error identifier with S.E. and F.

4. Identify all successful cases with error E.

5. Repeat steps 2-4.

This procedure will identify all known single errors which caused the failures. Multiple errors are identified by correcting two or more errors at a time in the program in state S.

Implicit in the above discussion is what appears to be previous knowledge of all errors. In practice, the programmers were given software failures from initial executions of the experiment driver. They then identified the associated errors and made their corrections. In some cases no corrections were made. These errors are discussed in Sections 4.3.6, 4.4.6, and 4.5.6.

3.3 COMPUTER/LANGUAGE ENVIRONMENT

3.3.1 BITS System

BITS is an integrated system of intelligent terminal hardware and software developed by BCS. The hardware consists of a Terak 8510A microcomputer using the 16-bit LSI-11 microprocessor. Primary memory is 56K bytes, and secondary memory consists of two eight-inch floppy disk drives with a combined capacity of about 512K bytes.

The software available on BITS includes a file manager, text editor, communications capability, together with Pascal, BASIC and FORTRAN compilers. The FORTRAN compiler, utilized in this study, closely conforms to Standard ANSI FORTRAN 77.

3.3.2 VAX/VMS System

VAX (Virtual Address Extension) system together with VMS (Virtual Memory Operation System), is a high-performance multi-programming system based on 32-bit architecture. Virutal memory features, plus disk storage of over one gigabyte, remove the storage restrictions for most applications programs.

There is a full suite of software available on the VAX, including an operating system, text editors and compilers. The FORTRAN compiler contains all the features of Standard ANSI FORTRAN 77.

4.0 EXPERIMENT DATA COLLECTION

4.1 INTRODUCTION

This chapter and the referenced appendices present the data-collection results of the experiment. Also included is a brief description of the programmers' backgrounds.

There are three parallel sections (4.3, 4.4, and 4.5) corresponding to the three problems of the experiment. Each of these sections together with corresponding appendices gives, in detail, background information on the problem specifications and correct version, descriptions of test cases and the usage distribution for the experiment runs.

A tabulation of software errors is given for each subject program. The identified errors are categorized using the categories in [10]. These categories are given in Appendix A.

4.2 PROGRAMMER DESCRIPTIONS

Two scientific programmers were used for all three problems. Programmer A received a B.S. degree in Computer Science in 1979 and joined BCS in June 1979 as a programmer. His principal job has been to support and enhance a geometry package used to design wing and body configurations. His emphasis in the field of computer science is structured software design, development and languages, including FORTRAN, Pascal, ALGOL, SNOBOL and COBOL.

Programmer B received a B.S. degree in Computer Science in 1975 and joined BCS in January 1976 as a programmer. He has worked on nuclear-waste engineering and radiation-monitoring problems, using FORTRAN on a variety of machines. Later assignments have involved integration testing on the AWACS program using JOVIAL language, and conversion of a missile-simulation program and graphics package from an IBM machine to the VAX/VMS system.

4.3 PROBLEM #1

4.3.1 Background

TRW [10] conducted an extensive study of software reliability in 1973 which included an experiment similar to that of the present study. Two experienced programmers were given specifications for a missile-tracking simulation problem. Then each programmer designed, coded and tested his own version using FORTRAN IV.

The experiment consisted of many executions of the resulting programs, using a predefined usage distribution. Reliability estimates were made from the percentage of successful cases to the total number of cases run. However, there was no error-correction process as that in this study.

The same problem specifications, test cases and usage distribution in the TRW study were used in the present study. However, additional specifications, including input and output descriptions, were necessary to resolve certain ambiguities.

4.3.2 Specifications

In general, the specifications require geometric calculations involving an input set of two-dimensional coordinates representing radar tracks. Fifteen such calculations then became conditions to be logically combined to determine if a make-believe missile should be launched.

The complete specifications for problem #1 are presented in Appendix B. Sections 1.0 - 4.0 of Appendix B are from [10]. Sections 5.0 - 7.0 of Appendix B were added as part of the present study.

4.3.3 Test Cases

Three test cases were used to bring both subject programs A1 and B1 to state 0. All three cases were chosen from [10]. The test cases and the corresponding correct output are presented in Appendix C.

4.3.4 Usage Distribution

The following usage distribution (see 2.1.2) describes the input data used for problem #1.

4.3.4.1 Coordinates

Figure 4.3.4.1-1 illustrates the distribution from which the two-dimensional coordinates simulating a radar track are drawn. Each input case requires 5 such coordinates.

Approximately 95% of the time the two-dimensional coordinates are drawn uniformly from area A, and 5% of the time from the combined 4 subareas labeled B. Each coordinate is rounded to the nearest 0.1.

4.3.4.2 LCM and PUM Elements

120 elements of the Logical Connector Matrix (LCM) satisfying L_{ij} such that $i \leq j$ (where i is the row index and j the column index) were generated from the following distributions:

for i and j equal to 1,2,3,4,5,6,7,8,9,10,11 and $i \leq j$, $L_{ij} = 2$

for all other i and j where $i \leq j$,

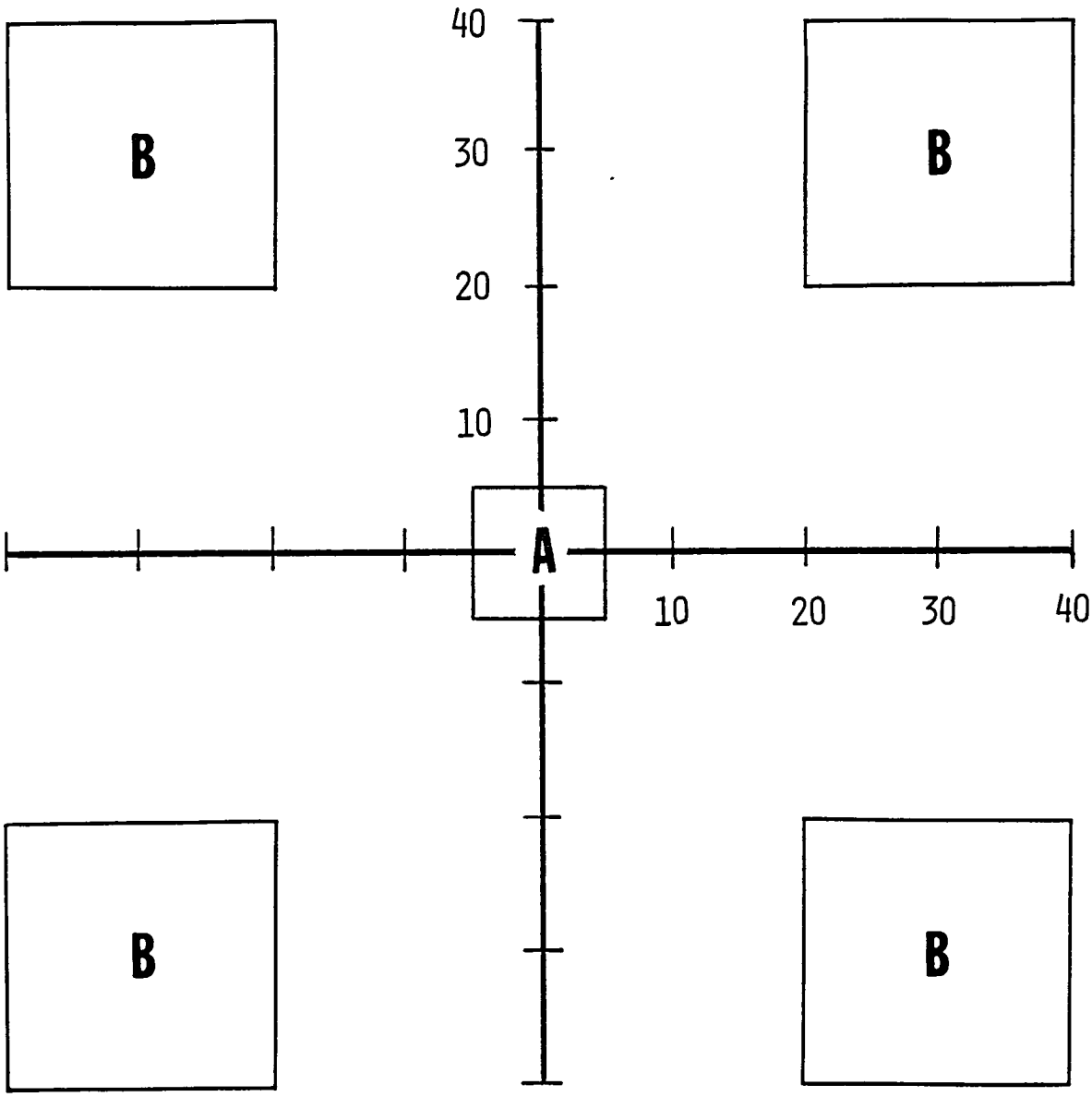
$$Pr\{L_{ij} = 0\} = .78$$

$$Pr\{L_{ij} = 1\} = .20$$

$$Pr\{L_{ij} = 2\} = .02$$

the L_{ij} for which $i \geq j$ were computed internally from the equation $L_{ij} = L_{ji}$.

Fifteen diagonal elements, P_{ii}, of the Preliminary Unlocking Matrix (PUM) were generated by selecting 0 or 1, each with probability 0.5.



P(A) = 95%, uniformly distributed within A
P(B) = 5%, uniformly distributed within B

Figure 4.3.4.1-1. Usage Distribution for Coordinates.

4.3.4.3 Parameters

The 15 Launch Interceptor Conditions (LIC) required the following parameters (listed by the appropriate LIC):

1) ℓ = 14.5
2) r = 7.1
3) ϵ_2 = 0
4) A = 50
5) M = 1; Q = 3
6) None
7) N = 4; ϵ_1 = 15.0
8-11) n_1 = $n_2 = n_3 = n_4 = 1$; $m_1 = m_2 = m_3 = m_4 = 1$
12) n_6 = 5
13) L = 0
14) R = 0
15) E = 0

4.3.5 Correct Version

Correctness of output was determined by comparing the subject programs' output with that from one program written in the TRW study.

Program "MYRON" was written by a senior-level applications programmer, as part of the TRW project. The program consisted of approximately 298 lines of non-comment source code of FORTRAN IV.

After program MYRON passed the three test cases in Appendix C, 1000 randomly generated input cases from the usage distribution of Section 4.3.4 were executed with program MYRON. Based on three software failures among these cases, the reliability of program MYRON was estimated to be 0.997.

As part of the present study, the software error causing the three failures above was corrected. The resulting program was then assumed to be correct. However, in the course of the experiment two additional software errors in MYRON were corrected. After the corrections were made, the experiment was rerun with what is now assumed the correct version.

4.3.6 Error Descriptions

In the course of the experiment for problem #1, subject program A1 had 9 different software failures detected and the corresponding errors corrected. An additional software error, manifested in small inaccuracies between A1 and the correct version, was left uncorrected. Subject program B1 had 9 different software errors corrected. All errors are discussed in the following sections.

4.3.6.1 Subject Program A1

ERROR NUMBER	CLASSIFICATION CODE	DESCRIPTION
1	A600	An incorrect algorithm was used for coverage of three coordinates by a circle of given radius.
2	A600	An incorrect equation was used to calculate the area of the triangle.
3	A600	The use of arccos to determine linearity of three points was inaccurate.
4	B600	An incorrect comparison was used when determining if the difference between abscissa of adjacent points were negative.
5	A600	There was a division by zero when coordinates were identical in the calculation of the area of the triangle defined by three coordinates.
6	B400	There was an accuracy failure in an algorithm for coverage of three coordinates by a circle of a given radius when two coordinates were identical.
7	B400	This error is identical to #6 except it is located in a separate, but independent section of the program.
8	A600	The tolerance introduced for error #3 had to be made larger. This is an example of the ripple effect, Section 5.7.
9	A600	The tolerance introduced for error #6 had to be made larger. (Ripple effect.)
10	A600	The tolerance introduced for error #8 was too large. No correction was made, because a new algorithm was probably required, not just a simple tolerance change. (Ripple effect.)

4.3.6.2 Subject Program B1

ERROR NUMBER	CLASSIFICATION CODE	DESCRIPTION
1	A400 D100*	Conversion to radians of an argument for arccos was redundant. In addition, pi was not initialized everywhere.
2	F300*	The squaring of a negative quantity was done using a power-series expansion, not just squaring the quantity, i.e., W**2.0 was used instead of W**2 or W*W.
3	A600*	In the algorithm for coverage of three coordinates by a circle of a given radius, a term was left out of an equation.
4	A600	There was a division by zero, when adjacent coordinates were the same.
5	A900	The argument for arccos was greater than 1.0 when three coordinates formed a line.
6	B400	An exact comparison of floating-point variables was used, instead of allowing a tolerance.
7	A600	An incorrect algorithm was used for LIC(7) when the first and last coordinates were identical.
8	A600 D100	The area of the triangle defined by three coordinates was not set to zero when the coordinates formed a line.
9	A900	When the area of the triangle defined by three coordinates was exactly equal to the test criteria (parameter A), a value slightly greater than A was calculated.

* Hidden errors, incorrectly identified as detectable.

4.3.7 Run Results

4.3.7.1 Subject Program A1

Figure 4.3.7.1-1 presents the results of the experiment for subject program A1. This figure, as well as those for the remaining programs (4.3.7.2-1, 4.4.7.1-1, 4.4.7.2-1, 4.5.7.1-1 and 4.5.7.2-1), traces the 50 runs for the particular program. The figure is composed of levels, or stages, of program states, where each stage is defined by the number of errors detected. Beginning with state 0, the occurring program states and their frequencies are shown for the 50 runs. The encircled number(s) represent a program state, in particular, the error numbers of the corrected errors. For example, 12 is a given subject program with errors #1 and #2 corrected. A subject program at state 0 is identified with 0 following the program name and dash, e.g., A1-0. The directed line segments represent the random walk of the subject program going from one state to another, i.e., having one or more errors corrected. The number to the left of this line segment is the number of runs experiencing that particular change in state.

For example, using Figure 4.3.7.1-1 and beginning with state 0 (A1-0), 40 of the runs had error #1, 8 had error #2 and 2 had multiple errors, #1 and #2, occurring with the same input cases. As shown in the table to the right of the figure, these 50 runs required a total of 51 input cases for the first error(s) to occur.

From these states (48 at stage 1, 2 at stage 2), the runs continue to another stage. Note that not all 50 runs continue through all stages, because some errors detected are not corrected. In general, the number of input cases per run required to detect errors increases as the stage increases.

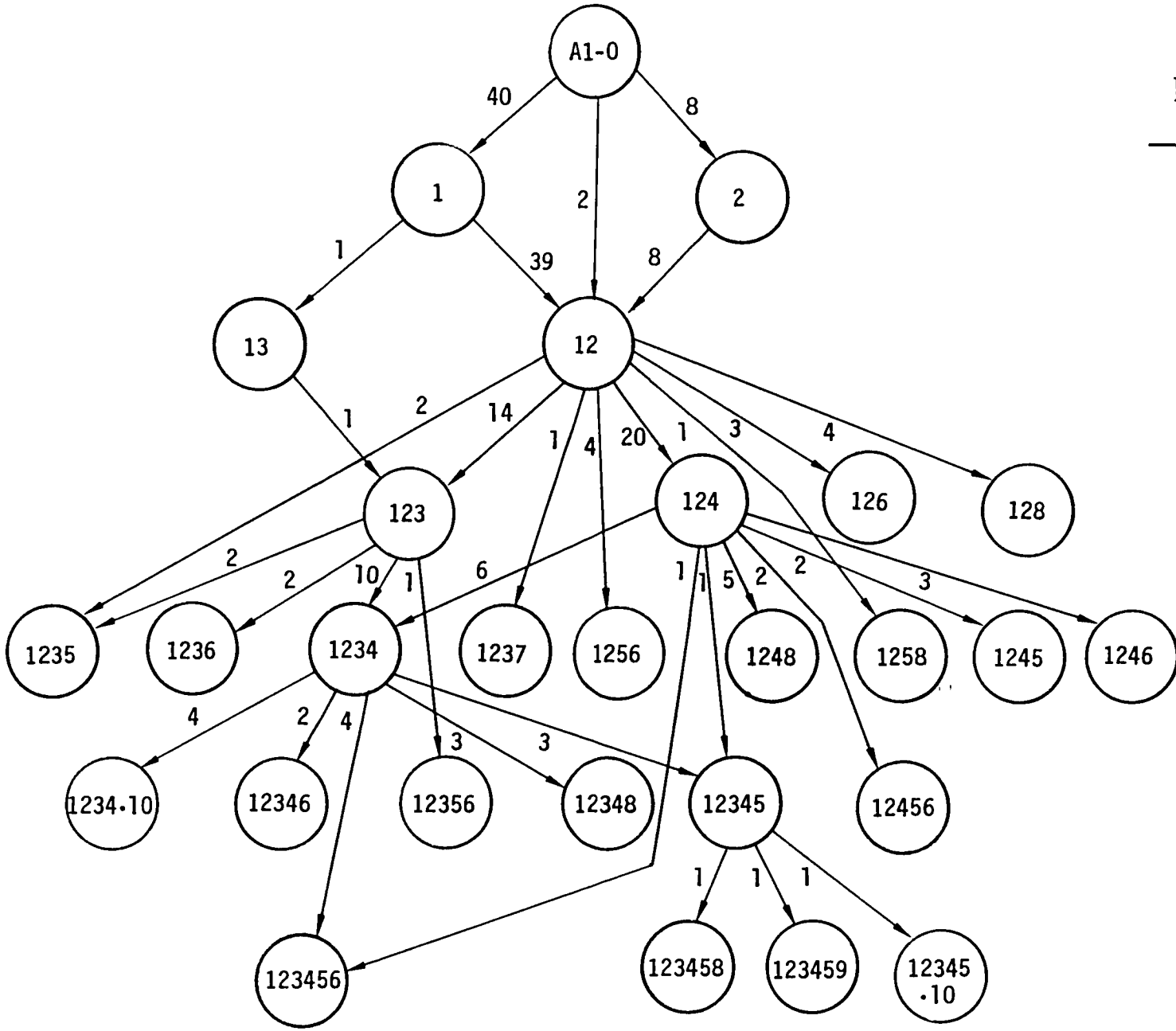
4.3.7.2 Subject Program B1

Figure 4.3.7.2-1 illustrates the results of the software failure detection/error correction process.

4.4 PROBLEM #2

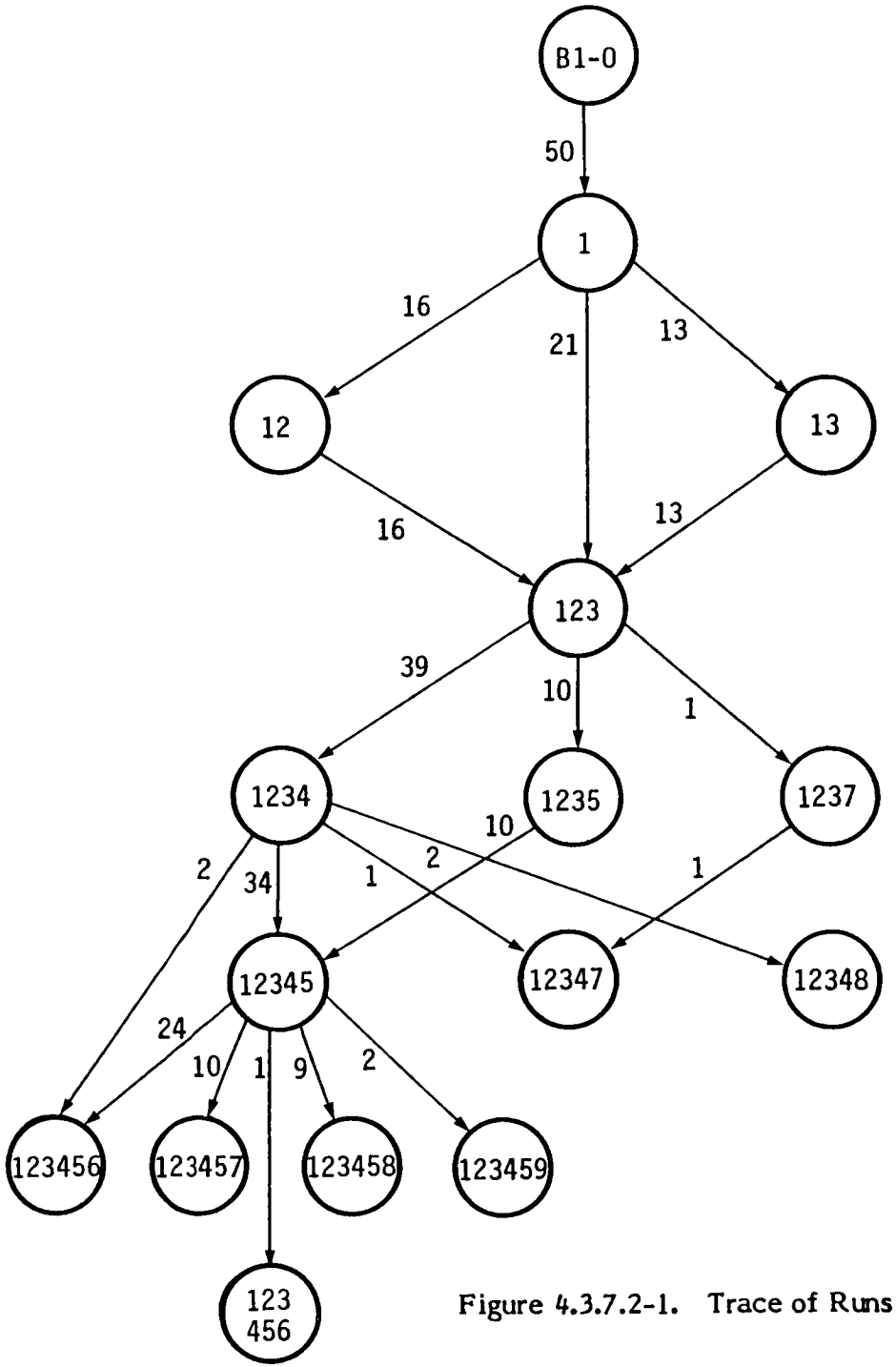
4.4.1 Background

The Boeing Intelligent Terminal System (BITS) includes a library of general-purpose scientific subroutines. Included in the library are routines for spline-function interpolation. These routines have received extensive testing and subsequent use at Boeing for over ten years. Functions of several of these routines were used for specifications of problem #2 and the library routines themselves became the basis of the correct version (see 4.4.5).



ERRORS DETECTED (STAGE)	RUNS	TOTAL CASES
1	50	51
2	50	466
3	50	18,835
4	43	18,316
5	19	17,586
6	8	2,480

Figure 4.3.7.1-1. Trace of Runs for Subject Program A1.



ERRORS DETECTED (STAGE)	RUNS	TOTAL CASES
1	50	328
2	50	107
3	50	89
4	50	5,484
5	50	14,355
6	46	81,595
7	1	0

Figure 4.3.7.2-1. Trace of Runs for Subject Program B1.

4.4.2 Specifications

Specifications for problem #2 were developed as part of the present study. In general, four subroutines were required: (1) routine to calculate coefficients of the spline passing through the input coordinates; (2) routine to calculate coefficients when the spline polynomials are expressed in an alternate form; (3) routine to interpolate at an arbitrary point using the spline coefficients, and (4) a routine to integrate along the spline between arbitrary endpoints.

The complete specifications for problem #2 are given in Appendix D.

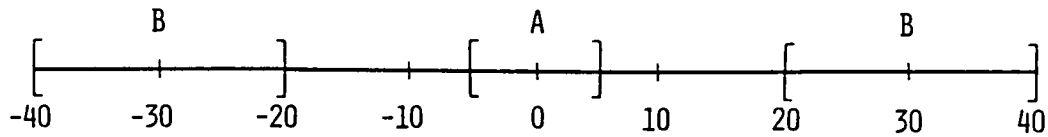
4.4.3 Test Case

One test case was used to bring both subject programs A2 and B2 to state 0. The test case and the corresponding correct output are presented in Appendix E.

4.4.4 Usage Distribution

Three coordinates are drawn randomly from the distribution shown in Figure 4.3.4.1-1. Each coordinate is rounded to the nearest 0.1.

Each input data case also requires three X-coordinates - one for the point of interpolation and two to define the limits of integration. These points are drawn from the following distribution:



P(A) = 95%, uniformly distributed within A
P(B) = 5%, uniformly distributed within B

Each of these X-coordinates is rounded to the nearest 0.1.

4.4.5 Correct Version

The correct results for three of the four required routines were those from the library routines based on which problem #2 was selected (see 4.4.1). The so-called correct results for the routine performing integration were computed in a routine written specifically for the project. This latter routine received extensive peer review and testing.

4.4.6 Error Descriptions

For problem #2, subject program A2 had five different software failures detected with four errors corrected. The fifth software failure was when a subject program output item was not within 1% relative error of the corresponding correct-version output item. In this case, there was no correction made to the software.

Subject program B2 had three different software failures, with two failures resulting in corresponding software-error corrections. One software failure was similar to the relative-error failure mentioned above. As with subject program A2, there was also no correction made for this failure in subject program B2.

4.4.6.1 Subject Program A2

ERROR NUMBER	CLASSIFICATION CODE	DESCRIPTION
1	B400 B600 D800	An infinite loop occurred because a switch of variables was not made in a sort algorithm.
2	A600	The upper limit of the range of integration was incorrectly calculated when both limits were within the range of one spline.
3	A600	The integral was incorrect when the upper limit equaled the largest X-coordinate defining the splines.
4	A600	The interpolated value was incorrect when the point for interpolation equaled the largest X-coordinate defining the splines.
5	A600	There was an accuracy failure in some output item (relative error > 1%).

4.4.6.2 Subject Program B2

ERROR NUMBER	CLASSIFICATION CODE	DESCRIPTION
1	A800	The integral was not calculated when the lower limit was greater than the upper limit.
2	A600	The integral was incorrect when the upper limit equaled the largest X-coordinate defining the splines.
3	A600	There was an accuracy failure in some output item (relative error > 1%).

4.4.7 Run Results

4.4.7.1 Subject Program A2

Figure 4.4.7.1-1 presents the results of the software failure detection/error correction process.

4.4.7.2 Subject Program B2

Figure 4.4.7.2-1 illustrates the results of the software failure detection/error correction process.

4.5 PROBLEM #3

4.5.1 Background

The use of problem #3 in the experiment was motivated by existing routines used for earth-satellite calculations. Such calculations, involving analytic geometry and vector analysis, determine distances, azimuths and intersections on the earth.

4.5.2 Specifications

Specifications for problem #3 were developed for the project by a senior-level computer scientist who has had experience with the particular application stated above. The complete specifications are presented in Appendix F.

4.5.3 Test Case

One test case was used to bring both subject programs A3 and B3 to state 0. The test case and the corresponding correct output are presented in Appendix G.

4.5.4 Usage Distribution

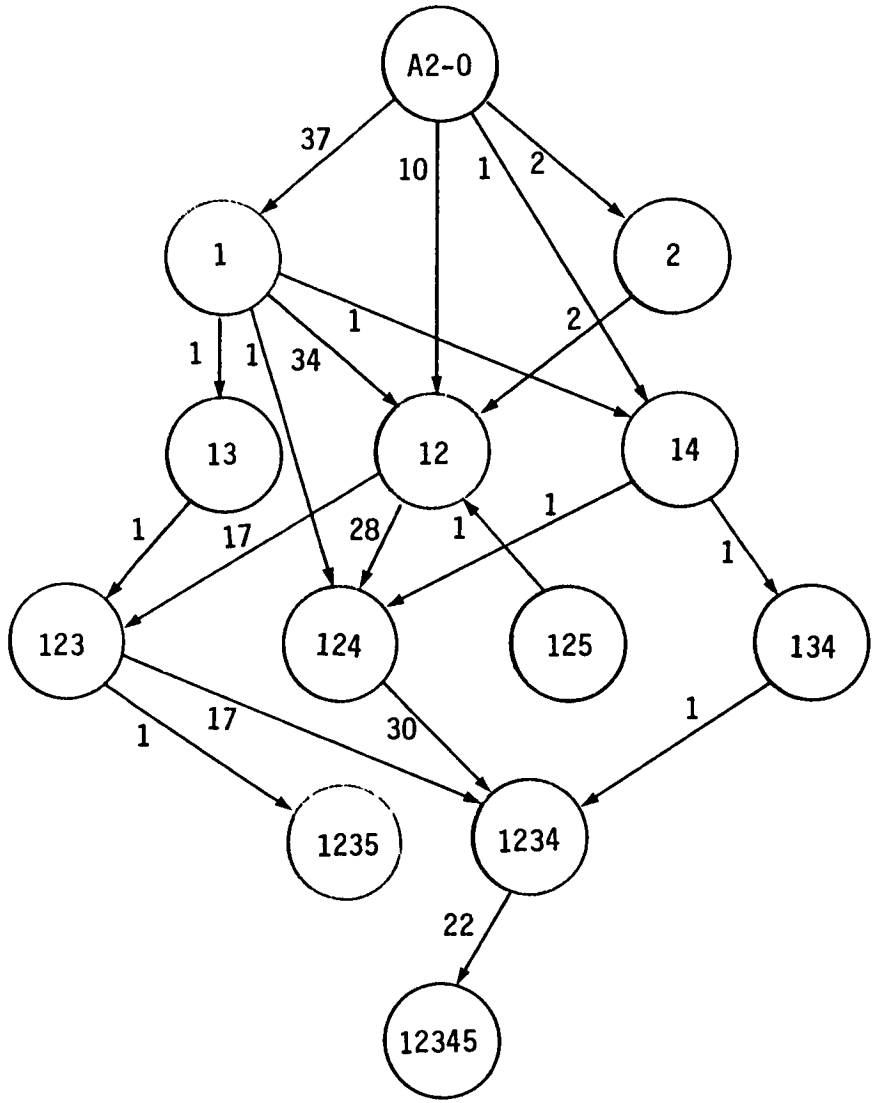
As stated in the specifications for problem #3 (Appendix F), three latitude-longitude coordinates on the earth are required inputs, as well as an angle between 0° and 180°. The distribution for the latitude-longitude coordinates was uniform over the sphere, but rounded to the nearest 5° in both latitude and longitude. The distribution for the angle was uniform between 0° and 180° with no rounding.

4.5.5 Correct Version

The existing routines mentioned in Section 4.5.1 became the nucleus of the "correct version" program which was designed, coded and tested for the study by the senior-level computer scientist who developed the specifications. His version received extensive peer review plus many tests designed to simulate the usage distribution to be used in the experiment.

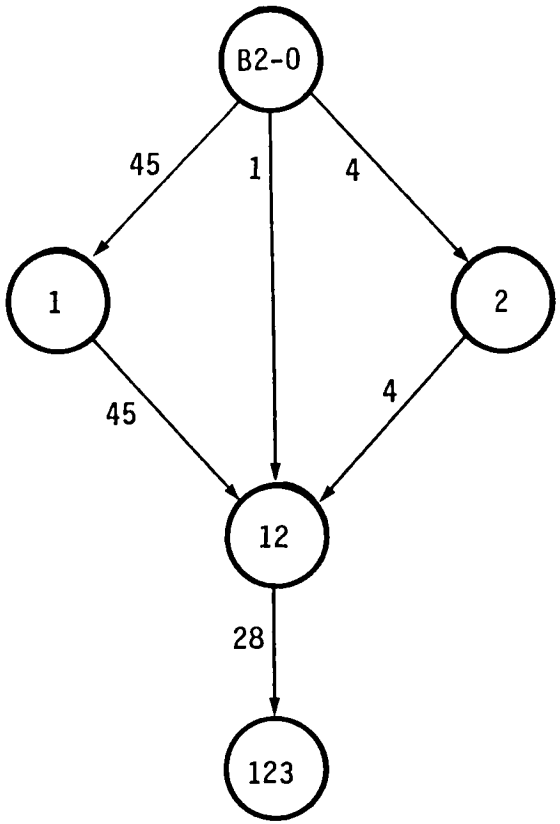
4.5.6 Error Descriptions

For problem #3, subject program A3 had seven different software failures, for which five distinct software errors were corrected. One failure for which no error was corrected involved the wrong number of intersections found (see Section 1.0,



ERRORS DETECTED (STAGE)	RUNS	TOTAL CASES
1	50	62
2	50	178
3	50	1,323
4	49	2,567
5	22	162,142

Figure 4.4.7.1-1. Trace of Runs for Subject Program A2.



ERRORS DETECTED (STAGE)	RUNS	TOTAL CASES
1	50	345
2	50	1,903
3	28	259,110

Figure 4.4.7.2-1. Trace of Runs for Subject Program B2.

Appendix F). The second such failure occurred when the relative error of an output item from A3 compared to the correct version was greater than 1%. Time and budget constraints precluded software error corrections for these two failures.

Similarly, subject program B3 had ten different software failures for which six software errors were corrected. Two of the remaining software failures which did not lead to corrections correspond to similar failures for program A3 above, with an additional two failures involving a division by zero and an incorrect azimuth.

4.5.6.1 Subject Program A3

ERROR NUMBER	CLASSIFICATION CODE	DESCRIPTION
1	A600	The determination of the sign of the azimuth was incorrect.
2	A800	There were uninitialized variables when cross-product calculations were bypassed under certain conditions.
3	A900	The argument for arccos was greater than 1.0 or less than -1.0.
4	A600	The algorithm to determine intersections calculated the wrong point of intersection.
5	A600	The azimuth was incorrectly calculated when the path went through either the north or south pole.
6	A600	The algorithm for calculating intersections failed to determine the correct number of intersections.
7	A600	There was an accuracy failure in some output item (relative error > 1%).

4.5.6.2 Subject Program B3

ERROR NUMBER	CLASSIFICATION CODE	DESCRIPTION
1	A800	The determination of the sign of the azimuth was not done.
2	A600	The algorithm to determine the order of the two intersection points was incorrect.

ERROR NUMBER	CLASSIFICATION CODE	DESCRIPTION
3	A900	The argument for arccos was greater than 1.0 or less than -1.0.
4	A600	The algorithm to determine intersections failed to find a correct intersection point.
5	A900	The argument for arcsin and/or arccos was greater than 1.0 or less than -1.0.
6	A600	The sign of the calculated azimuth was incorrect, when the magnitude of the azimuth is pi.
7	A600	Determination of colinearity of two coordinates and the center of the earth was incorrect.
8	A600	There was an accuracy failure in some output item (relative error > 1%).
9	A600	There was division by zero when determining intersections.
10	A600	The azimuth was incorrectly calculated as 0, when the correct value was pi.

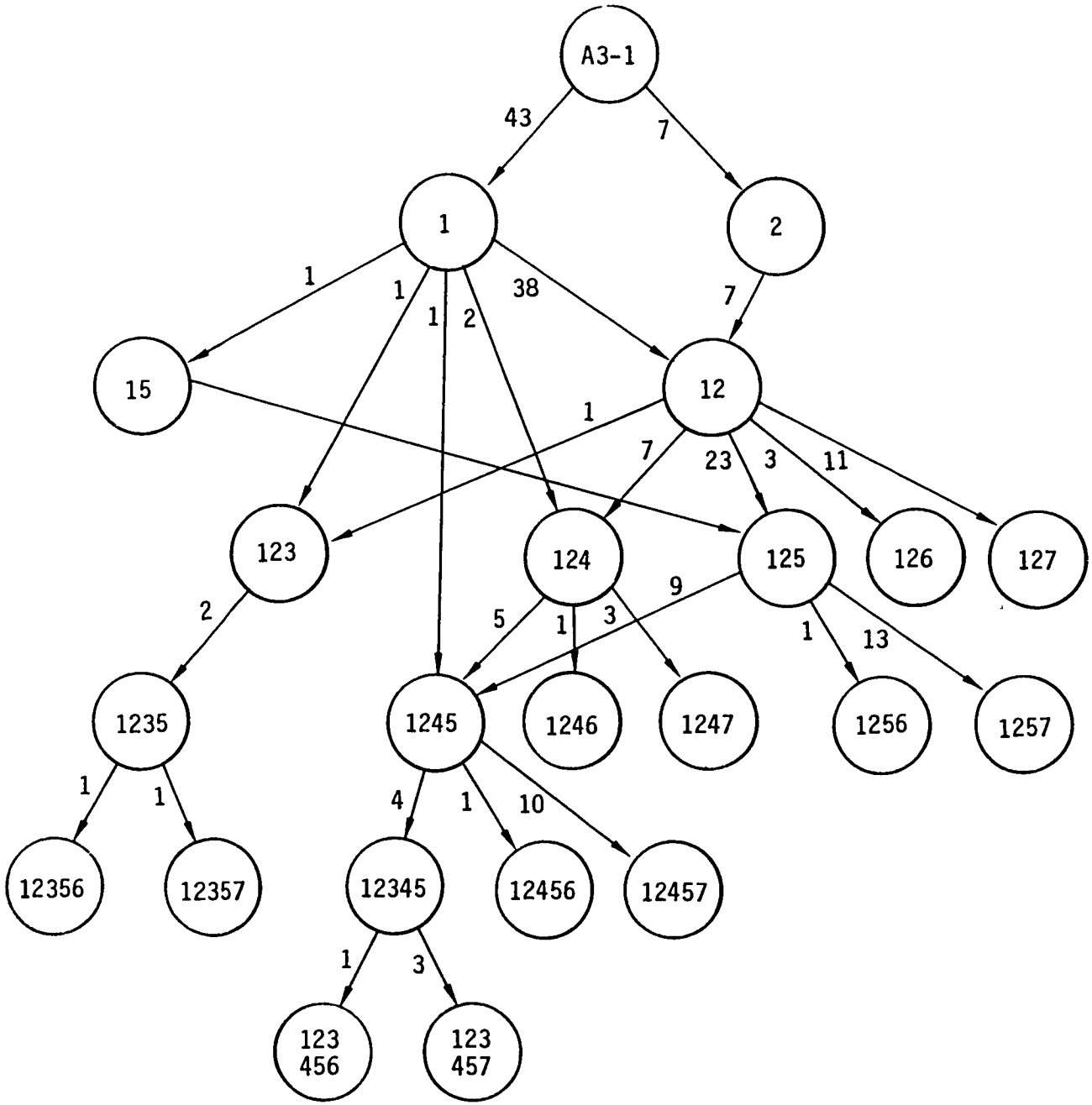
4.5.7 Run Results

4.5.7.1 Subject Program A3

The results of the software failure detection/error correction process are shown in Figure 4.5.7.1-1.

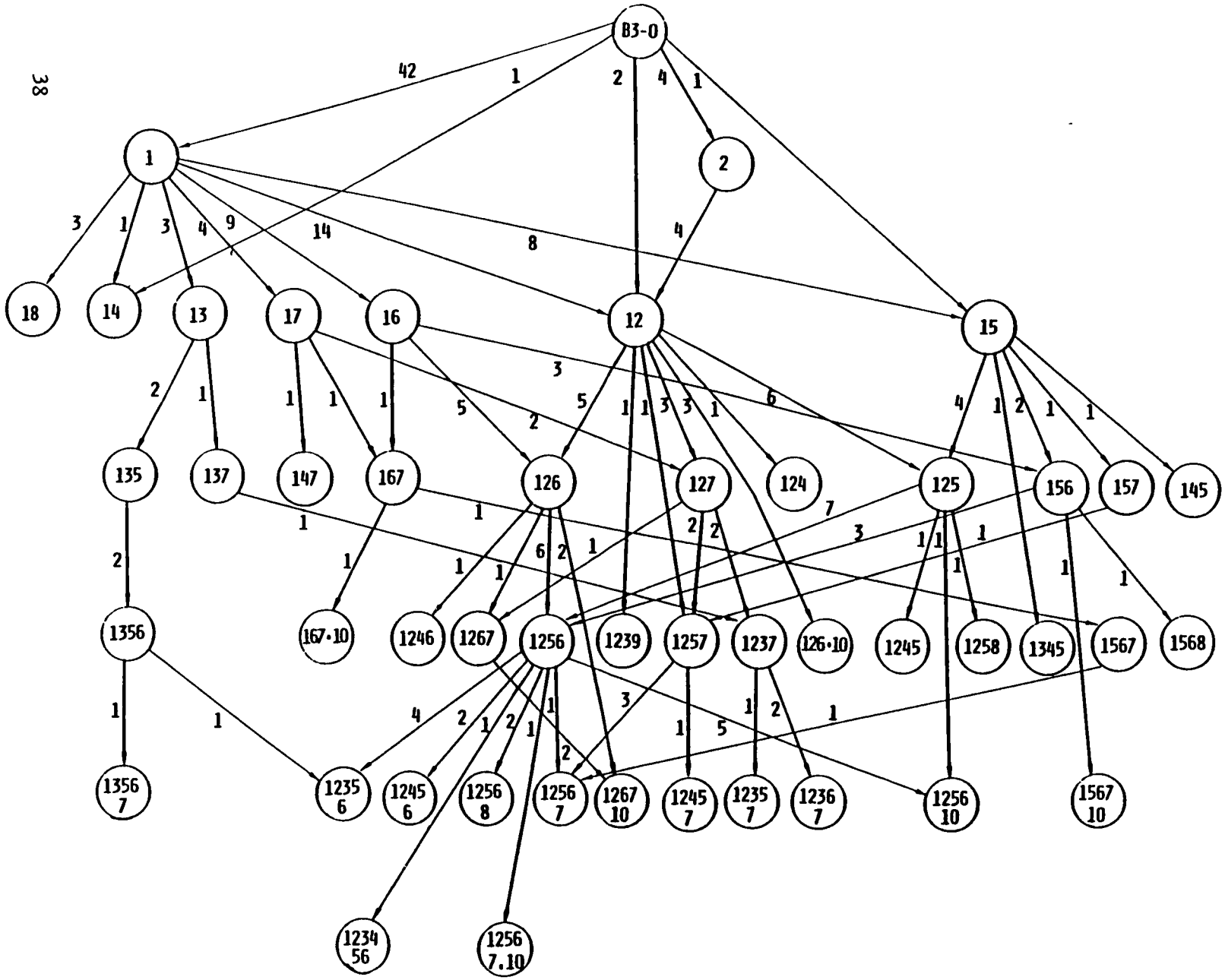
4.5.7.2 Subject Program B3

The results of the failure detection/error correction process are illustrated in Figure 4.5.7.2-1.



ERRORS DETECTED (STAGE)	RUNS	TOTAL CASES
1	50	137
2	50	482
3	49	4,875
4	35	7,690
5	17	7,050
6	4	3,220

Figure 4.5.7.1-1. Trace of Runs for Subject Program A3.



ERRORS DETECTED	RUNS	TOTAL CASES
1	50	201
2	50	899
3	50	1,277
4	42	1,484
5	32	1,226
6	2	0

Figure 4.5.7.2-1. Trace of Runs for Subject Program B3.

5.0 DATA ANALYSIS

The results contained in the following sections depend on the structure of the experiment performed. In particular, they reflect the specific features of the usage distributions employed, the detectors imposed to define the errors, and the sampling techniques utilized during experimentation.

The data base obtained from this experiment and on which the following analyses are conducted is presented in Appendices H-M. For each program the listing consists of run number and for each stage the number of executions to failure, the number of the error causing failure and the interfailure time.

One difficulty with this data should be noted. At the beginning of the experiment, the concept of how an error should be strictly defined was not completely understood. After some experience it was finally established that only those program errors causing the specific program failure would be corrected between states. In addition, it was also established that errors would be separately defined, i.e., labeled separately, only if for some input cases one would occur without the other. In other words two errors would be separately labeled if the intersection of the two error sets was strictly contained in, and not equal to, their union. Thus, errors that always occur together are defined as a single error and are removed together.

Unfortunately, the realization of this difficulty did not occur early enough to prevent a slight degradation of the data base. Specifically the distortion occurred for errors #2 and #3 of subject program B1, the first program to be studied. At the time of the debugging of error #1, programmer B was allowed to fix another line of code. This line of code had two intriguing properties. First of all, because of the way the computer was initialized and the specifications of the problem, this line of code, though incorrect, would not have resulted in a computational error, and thus falls in the class of hidden errors with respect to the set of error detectors utilized in this problem. Secondly, correcting this line of code permitted entry into a section of the subroutine containing errors #2 and #3. The code in this part of the subroutine, however, was not essential to the solution of the problem as specified, and as long as it was bypassed made no difference. Thus, errors #2 and #3 were hidden by the existence of the initial hidden error.

5.1 TESTING THE HYPOTHESIS OF EXPONENTIAL INTERFAILURE TIME

One of the major assumptions of most reliability models is that the conditional distribution of interfailure time given the number of errors corrected is exponentially distributed. When all errors are identically distributed or if the conditioning is on state rather than stage, this assumption appears consistent with the error detection process as commonly understood. If measured execution time, however, is highly truncated or widely varies with input, time may have to be measured in numbers of executions to failure to observe exponential behavior.

When errors are not identically distributed, interfailure time, conditioned on the number of errors corrected, is not necessarily exponentially distributed. In fact, if state conditioning produces exponentially distributed interarrival times, then stage conditioning results in times which are distributed according to a mixture of exponentials having one component for each state contributing to the stage. Thus,

for example, if there are three failure modes for a given program and if t is the interfailure time to the second failure given that one error has been detected and corrected, then t has density:

$$f(t) = \left\{ \lambda_1 (\lambda_2 + \lambda_3) \exp\left[-(\lambda_2 + \lambda_3)\, t\right] + \lambda_2 (\lambda_1 + \lambda_3) \cdot \exp\left[-(\lambda_1 + \lambda_3)\, t\right] + \lambda_3 (\lambda_1 + \lambda_2) \cdot \exp\left[-(\lambda_1 + \lambda_2)\, t\right] \right\} / (\lambda_1 + \lambda_2 + \lambda_3)$$

where λ_i is the failure rate of the i^{th} error source.

In general, a mixture of exponentials is a monotone decreasing function and is difficult to distinguish from the exponential itself. Its hazard function is a decreasing function. Table 5.1-1 compares values of the function f(t), above, when $\lambda_1 = 5, \lambda_2 = 10, \lambda_3 = 15$ with values of a comparable exponential function having the same first moment.

Some statistical goodness-of-fit tests have been performed on the data to establish if the exponential assumption is preferable against various alternative classes. Unfortunately none of these tests are sensitive enough to distinguish between an exponential and an exponential mixture. The tests performed were variations of the Kolmogoroff-Smirnov goodness-of-fit test developed specifically for the exponential with unknown parameter by Finkelstein and Schafer [11] and by Lilliefors [12]. Both tests are described in Mann, Schafer and Singpurwalla [13].

Table 5.1-2 gives the results of the test. In all cases, regardless of the data type, the exponential assumption is preferable to the alternatives to which these tests are sensitive. It does appear, however, that when few errors have been corrected in the early program stages this may not be the case due to the truncation effects of execution time.

Proschan's test for DFR distributions, Barlow and Proschan [14], could also be used in this context. This test appears to be particularly powerful in sensing differences coming from mixtures of exponential distributions. To date the computations in this test have not been completed.

For the problems considered in this experiment, constant execution time is a reasonable assumption. For problems demonstrating widely varying execution times or when there are sizable truncation effects, the equivalency of time and number of program executions may not hold.

Kalbfleish and Prentice [15] point out that if a data set is exponential, the log of the survivor function estimate plotted against t should approximate a straight line through the origin. The survivor function in the case of noncensored data is defined by

$$F(t_{(i)}) = 1 - i/n$$

TABLE 5.1-1
COMPARISON OF EXPONENTIAL-MIXTURE DENSITY WITH EXPONENTIAL DENSITY

$N = 3$

$\lambda_1 = 5, \lambda_2 = 10, \lambda_3 = 15$

t	Exponential Mixture	Exponential (Same Mean)
0	18.33	17.65
.01	15.16	14.79
.02	12.55	12.40
.03	10.41	10.39
.04	8.64	8.71
.05	7.19	7.30
.07	4.99	5.13
.10	2.92	3.02
.15	1.22	1.25
.2	.52	.52
.3	.10	.09

TABLE 5.1-2
LILLIEFORS K-S TEST STATISTIC
FOR THE EXPONENTIAL DISTRIBUTION

#-Of-Input-Cases Based

Subject Program	Stage	Statistic (Max δ)	d.f.	Significance (α = .05)
A1	2	.139	50	Not sig.
A1	3	.089	50	Not sig.
A1	4	.101	42	Not sig.
B1	4	.0602	50	Not sig.
B3	4	.0641	36	Not sig.

Execution-Time Based

Subject Program	Stage	Statistic (Max δ)	d.f.	Significance (α = .05)
A1	3	.0892	50	Not sig.

Figure 5.1-1 plots these quantities for two different stages and problems. There appears to be some deviations from the exponential in the tail. Whether this is due to real effects coming from a non-constant hazard function or random effects due to the variation in the tail is not known.

5.2 UNEQUAL ERROR PROBABILITY HYPOTHESIS

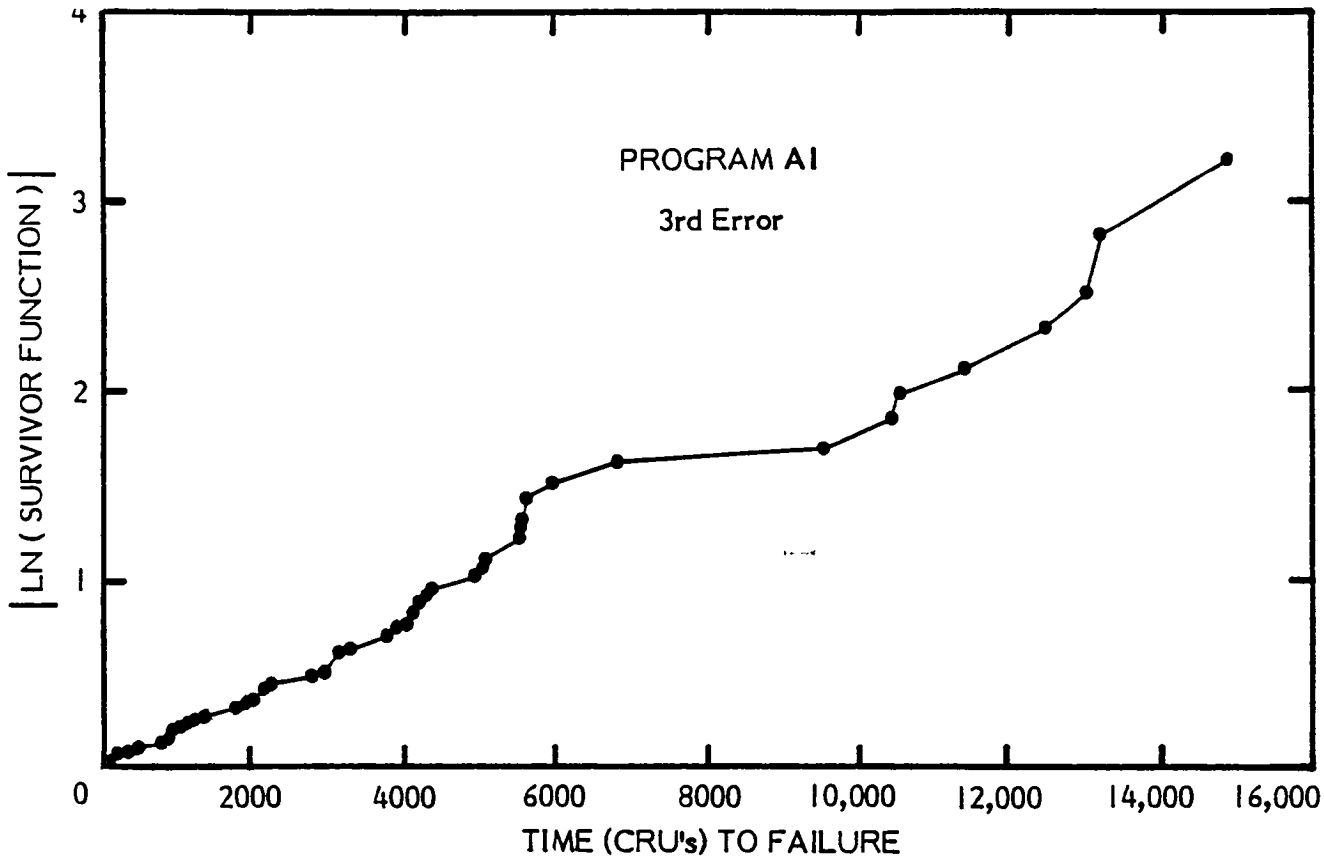
One of the primary concerns of this research is to determine if errors occur with unequal probabilities within a program. Estimates of these errors by number are given in Table 5.2-1 for all six subject programs. The estimates are based on looking at each run not as a series of stages but as a continuous uninterrupted run until the error of interest occurs. The number of executions until the error is manifested is distributed as the geometric distribution. The same is true for all other errors in the program. Thus the individual error estimates are based on the same estimator as that discussed in Section 2.2.9.

To compare two error estimates from the same run and statistically test for equality the following procedure was adopted. The numbers of correct cases to occurrence of each of the errors were subtracted run by run obtaining a set of k differences. It was then assumed that the mean of these differences was approximately normally distributed with mean zero if the errors are identically distributed. A t statistic was then used to evaluate if the observed mean difference was significantly different from zero, based on the observed standard deviation. For every program there were at least two errors which manifested a significant difference. Indeed by looking at the magnitude of the range of the error estimates, this hypothesis appears to be extremely well substantiated. In fact, it seems very unlikely that errors are ever identically distributed across an entire program error set.

5.3 STATE PROBABILITY ESTIMATES

Given that the above hypothesis is true, it is of great interest to measure its impact on the observed error structure of a program in a variety of situations. Table 5.3-1 gives estimates of the stage probabilities for each of the six programs of this experiment. The probability given for the i^{th} stage can be interpreted as the conditional probability that the program will fail in a single-execution given that i-1 errors have been corrected. This statement presumes then that the first stage is based on a program in its initial state as defined by the static detectors and that this is the state of the program with all of its errors intact.

Figure 5.3-1 is a plot of the absolute value of the logarithm of the error probabilities versus an approximate measure of the number of errors corrected at that stage. This graph presents the first experimental measure of a program's behavior through time, under repeated conditions, taking into account both the probabilities of the errors in the error set and their random order of observation. Several features of this graph are notable.



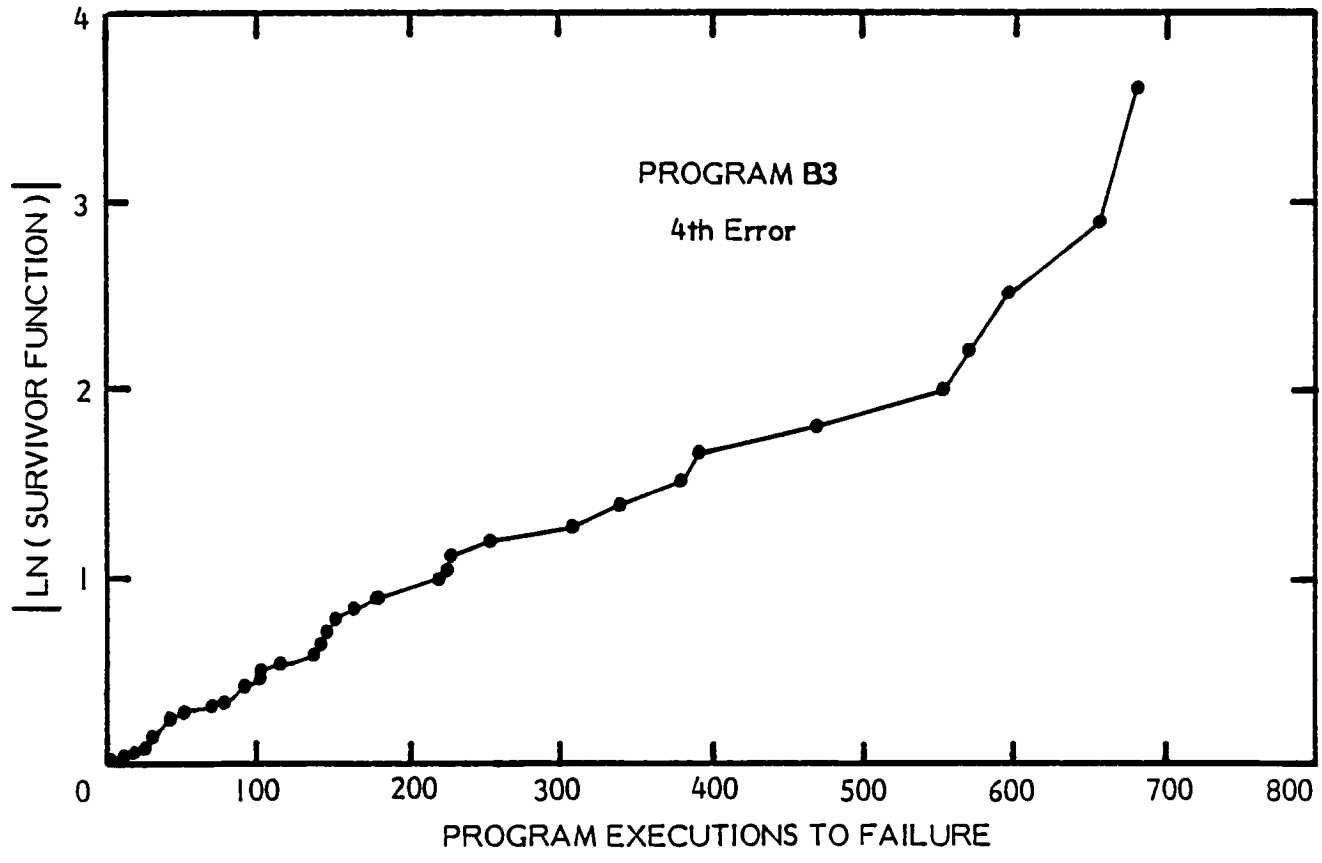


Figure 5.1-1. Survivor Function vs. t for Subject Programs A1 and B3.

TABLE 5.2-1
SPECIFIC ERROR PROBABILITIES - RANKED ESTIMATES

Program	Error No.	Prob. of Occurrence Per Program Execution	Program	Error No.	Prob. of Occurrence Per Program Execution
A1	1	8.20×10^{-1}	B1	1	1.52×10^{-1}
	2	9.63×10^{-2}		2	1.07×10^{-1}
	4	1.22×10^{-3}		3	1.02×10^{-1}
	3	8.17×10^{-4}		4	6.89×10^{-3}
	5	4.43×10^{-4}		5	2.41×10^{-3}
	6	4.12×10^{-4}		6	2.46×10^{-4}
	8	2.49×10^{-4}		7	1.18×10^{-4}
	10	1.15×10^{-4}		8	1.18×10^{-4}
	7	1.92×10^{-5}			
	9	1.92×10^{-5}			
A2	1	7.83×10^{-1}	B2	1	1.31×10^{-1}
	2	1.87×10^{-1}		2	2.26×10^{-2}
	3	1.72×10^{-2}		3	1.03×10^{-4}
	4	1.66×10^{-2}			
	5	1.44×10^{-4}			
A3	1	2.37×10^{-1}	B3	1	3.29×10^{-1}
	2	1.78×10^{-2}		2	8.13×10^{-2}
	6	1.16×10^{-2}		5	4.55×10^{-3}
	5	1.03×10^{-2}		7	1.75×10^{-3}
	7	4.08×10^{-3}		4	1.53×10^{-3}
	10	3.15×10^{-3}		6	3.84×10^{-4}
	3	2.51×10^{-3}		3	3.08×10^{-4}
	4	2.37×10^{-3}			
	8	1.38×10^{-3}			
	9	1.97×10^{-4}			

TABLE 5.3-1
ESTIMATED STAGE PROBABILITIES

PROGRAM	STAGE	PROB. OF ERROR PER EXECUTION
A1	1	.9803
	2	.1068
	3	.002602
	4	.002104
	5	.001176
	6	.0007659
B1	1	.1524
	2	.4673
	3	.4098
	4	.009117
	5	.003483
	6	.0005359
A2	1	.8065
	2	.2632
	3	.03759
	4	.01909
	5	.0001374
B2	1	.1449
	2	.02625
	3	.0001033
A3	1	.2488
	2	.05376
	3	.03524
	4	.02691
	5	.02302
B3	1	.3650
	2	.1037
	3	.01021
	4	.004681
	5	.002411

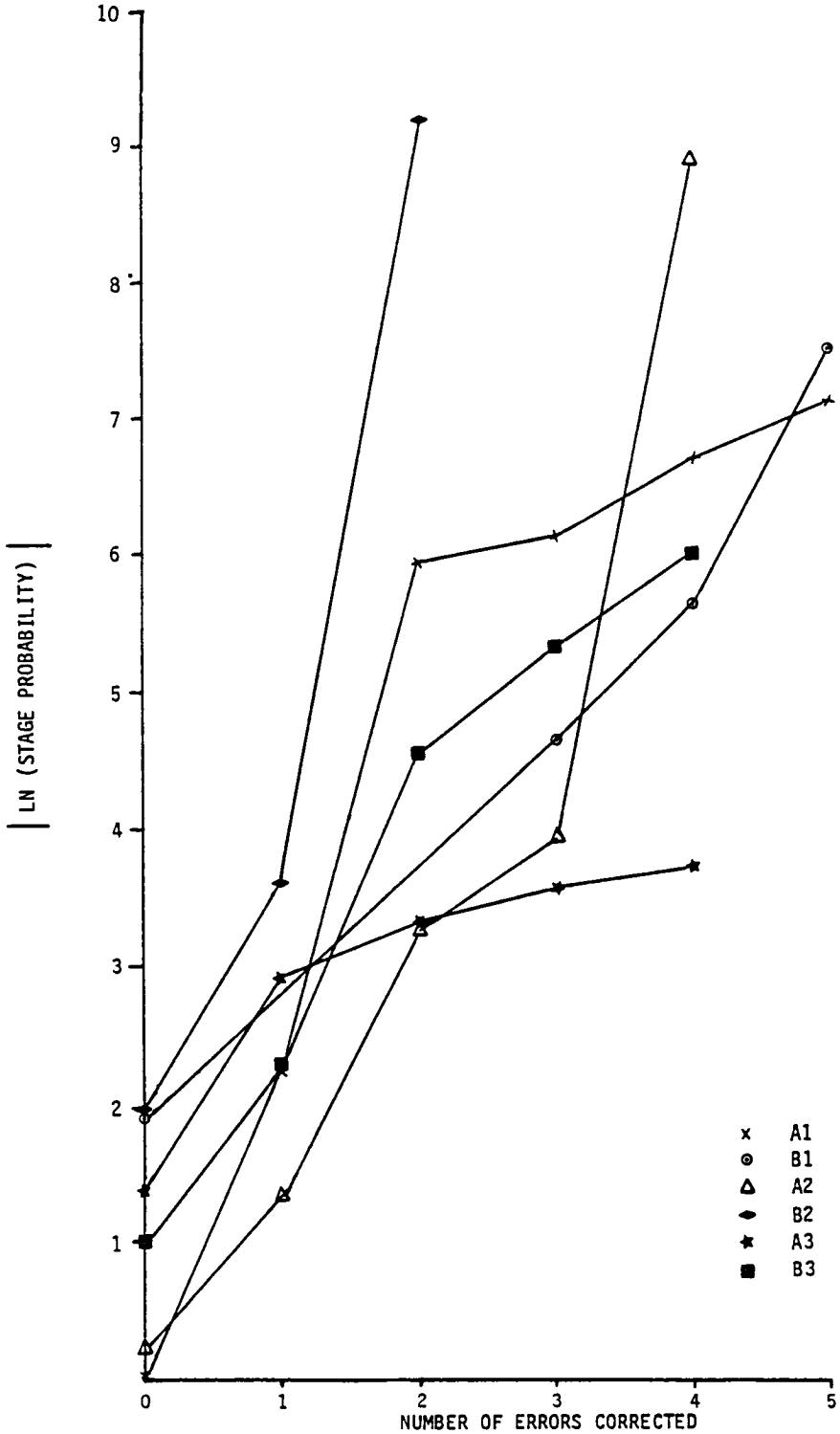


Figure 5.3-1. Estimated Error Rate as a Function of Errors Corrected, Using Original Data.

First of all, although controls were imposed on the experiment to insure as much commonality as possible in the definition of the initial program stage, this was an a priori definition. In fact what seems more desirable is a definition of the first stage that may be data dependent but that causes error probabilities across all programs to cluster. This is equivalent to defining the initial stage by a program feature that implies a certain state of probabilistic commonality. High error probabilities imply that a program is not well checked out. Check out is also very difficult to control in as much as programming time and programmer attention interact at this end of the scale with error discovery. Therefore, since highly probable errors are not interesting in forecasting software reliability, a redefinition of the initial program state in terms of a minimum p level might be potentially useful. This level was chosen to be ln p= -1.0 or p=.37. Figure 5.3-2 is a replot of the data in 5.3-1 with this definition of zero on the horizontal scale. The data is nicely compressed at zero with small range indicating some commonality. The usefulness of this definition is further explored in the next section.

One of the most striking characteristics of the data in Figures 5.3-1 and 5.3-2 regardless of the question of origin is the degree of linearity in each of the graphs. This suggests the model for the basic reliability structure behind these programs that is developed in the following section. In these graphs there is also a hint of a connection between the probability structure and the programmer as well as between the structure and the problem. The modeling effort referred to above enables these ideas to be subjected to statistical evaluation to some degree, and it also provides a framework in which to test program attributes that may be indicators of unreliability.

Another point about these graphs worth noting but of seeming little value to the current data analysis is the very low probabilities observed in what appears to be the early stages of programs A2 and B2. These two errors are both of the same type and both reflect a failure of the program to compute with the accuracy required by the problem specifications. It is not surprising then that such errors have similar probabilities in both programs.

5.4 RANDOM INFLUENCES ON THE STAGE PROBABILITIES

The randomness in the estimates of the stage probabilities has two sources. One source is due to the natural sampling variation in any exponentially sampled variable (or nearly exponential as in this case). This can be extensive for the exponential because the standard deviation equals the mean, and as the rate decreases it behaves more and more as a uniformly-distributed variable over an increasing range. The second source depends on the fact that the order of error detection can be random.

Figures 4.3.7.1-1 and 2, 4.4.7.1-1 and 2, and 4.5.7.1-1 and 2 have already demonstrated that program failures follow a random walk through the error space. The walk forms a pure death Markov process since the next error state of a program depends only on the transition probabilities and on those errors that have been corrected prior to execution and not on the time or order of correction. The frequencies of particular paths can be used to determine their transition probabilities. Most of the programs written for this experiment demonstrated a wide variety of walk behavior. Only for program B2 has the walk been concentrated primarily along a single path.

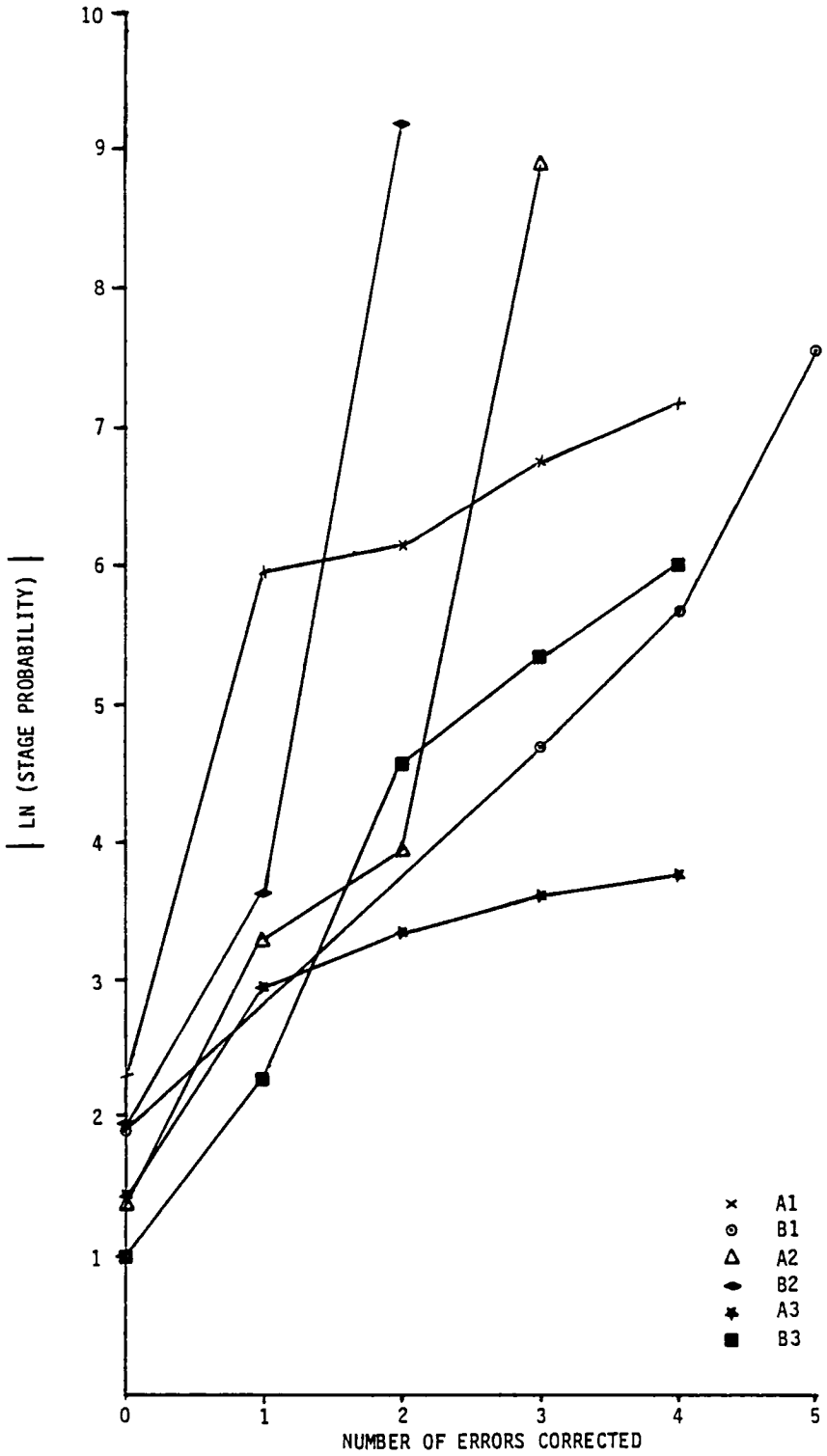


Figure 5.3-2. Estimated Error Rate as a Function of Errors Corrected, Using Modified-origin Data.

To evaluate the impact of these sources of randomness on the p estimates (which in turn are proportional to the failure rates when execution time is constant), histograms of the individual run estimates of p are given in Figures 5.4-1 and 5.4-2 for two of the programs. The sample mean and standard deviations are also given for each histogram in order to provide measures of the randomness in the failure rate estimates based on a single observation. Since single-point estimators are the current practice in traditional software reliability estimation, these histograms illustrate that the error in such observations can be considerable.

5.5 PROPOSED MODEL FOR SOFTWARE RELIABILITY BASED ON COX'S PROPORTIONAL HAZARDS FAILURE MODEL

Based on the evidence of Section 5.1, it can now be assumed that interfailure time is nearly exponentially distributed with rate proportional to p. In addition, the evidence in Figures 5.3-1 and 5.3-2 suggests that log p is nearly linear with respect to stage. A model that incorporates both of these ideas is explored in Kalbfleisch and Prentice [15] based on a model originally developed by Cox [16]. The model, called the proportional hazards model, specifies a hazard function of the form

$$\lambda(t;z) = \lambda(t)e^{z\beta}$$

where $\lambda(t)$ is the base line hazard function that may or may not be time dependent and where z is a vector of covariates or factors. Thus by providing multiplicative shocks to the hazard function the vector z can alter the rate at which an individual program proceeds along the time axis.

In the context of software reliability it is anticipated that z as a minimum should consist of a covariate representing the stage level, i.e., the number of corrected errors. In addition, the vector could include other covariates representing physical features of the program as well as the experimental test factors such as the problem and programmer factors. New covariates can be tested for their predicting ability as the model also provides a framework in which to statistically evaluate potential new explanatory covariates.

Since within a stage it has been demonstrated that the hazard rate is nearly constant, this implies that the model for software is much simpler than the model explored by Cox as $\lambda(t)$ must be constant. Under these circumstances then, the distribution of within- stage life length reduces to the exponential

$$f(t;z) = \lambda e^{z\beta} \exp(-\lambda e^{z\beta} t)$$

A program has been developed that provides maximum likelihood estimates for λ and β based on the Newton-Raphson optimization technique applied to this problem outlined in [15]. Two models have been explored using this program. Both are based on an expansion of the exponent in the hazard function in the following terms for the i^{th} problem and j^{th} programmer

$$\theta_{ij} = \log \lambda_{ij} + z\beta_{ij}$$
$$= \mu + \alpha_i + \beta_j + \gamma_{ij} k$$

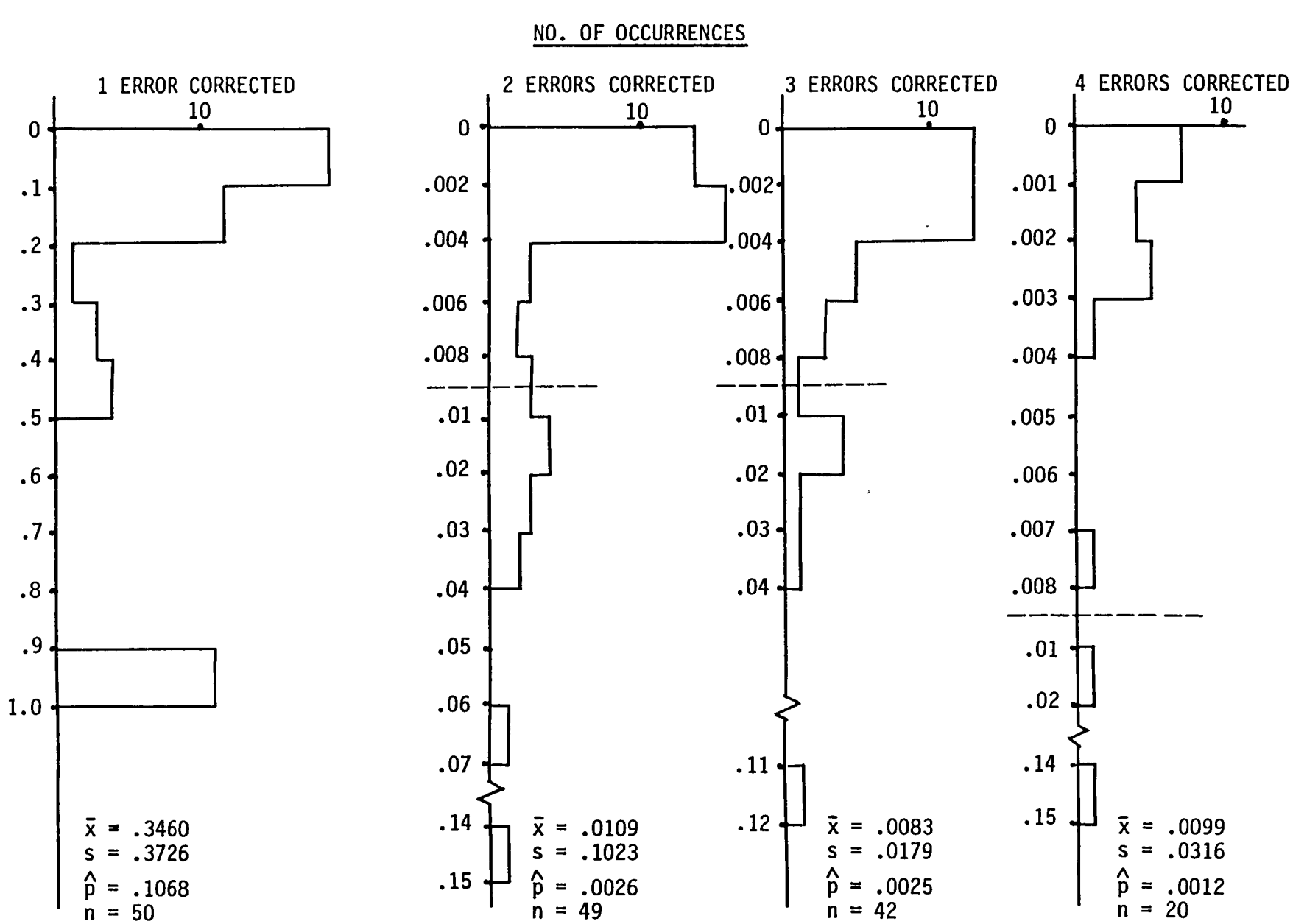


Figure 5.4-1. Histograms of Stage Probability Estimates for Subject Program A1 as a Function of the Number of Errors Corrected.

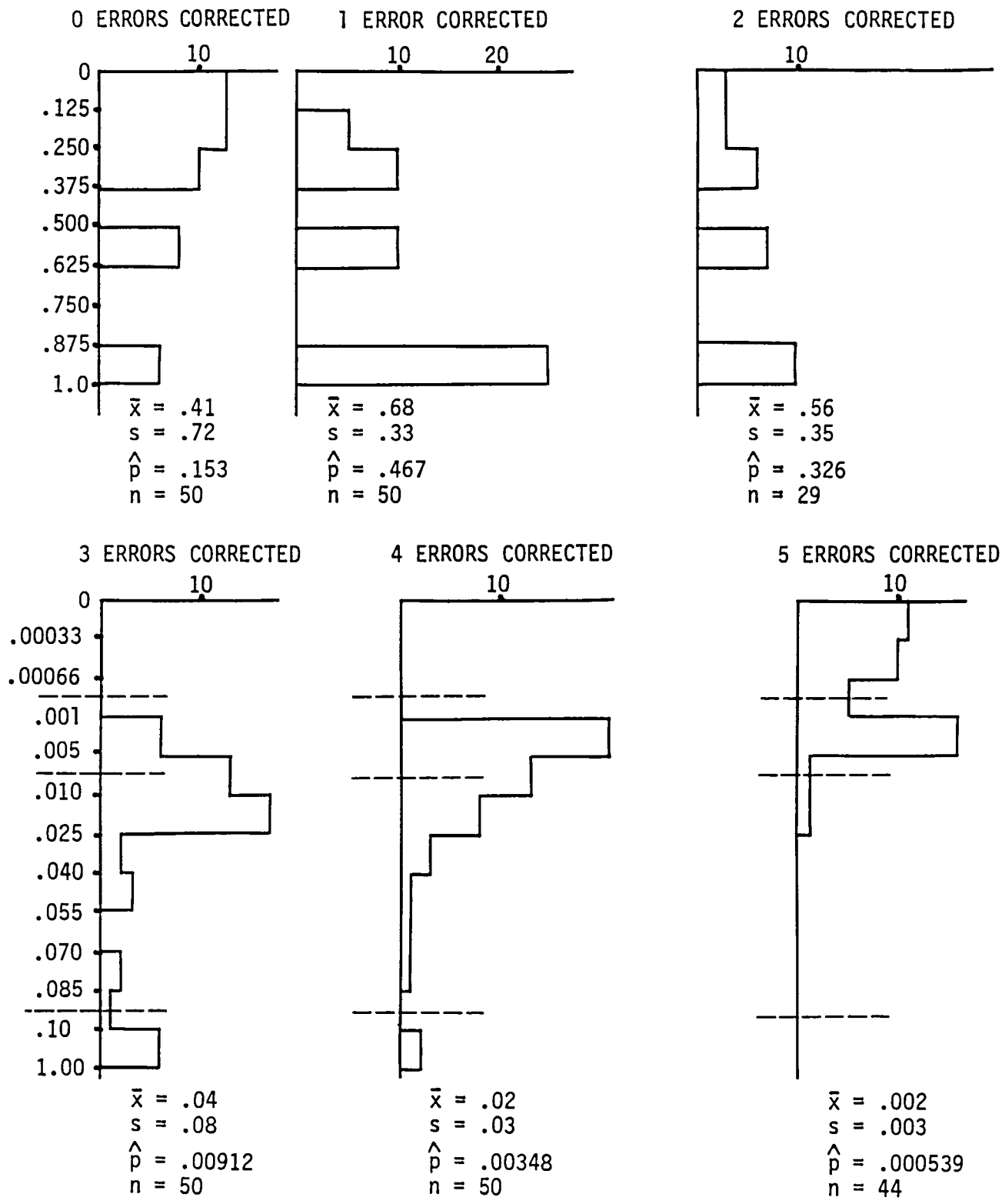


Figure 5.4-2. Histograms of Stage Probability Estimates for Subject Program B1 as a Function of the Number of Errors Corrected.

where

α_i = problem factor, i=1,2,3

β_j = programmer factor, j=1,2

γ_{ij} = program/programmer dependent slope

k = number of errors corrected, k=0,1,2,...

and

$\mu + \alpha_i + \beta_j$ = initial stage-one failure rate and background failure rate.

The data for the first model is based on the definition of the stage indicator illustrated by Figure 5.3-1 and the second model uses the definition of the indicator illustrated in Figure 5.3-2. Thus the two models differ only in the definition of k for programs A1 and A2. Model 2 modifies the definition and starts counting with k=0 only when $\ln p \leq -1$ for that stage and omits all prior error data. Model 1 on the other hand includes all of the observed data, and starts counting k=0 according to the original definition of a stage.

Because the problem with the data of program B1 was not completely understood at the time of the development of the proportional hazards model, the data from errors #2 and #3 was omitted. It now appears that the best compromise with this difficulty would be to ignore the existence of these errors, group their data with the data of error #4 as if they had not been recorded, relabel error #4 as #2 and relabel stages 3, 4, ... as stages 1, 2, ... The effect of these changes has not as yet been investigated.

The model for θ was chosen specifically as a first attempt to explain the nature of the dependence of the error structure on problem and programmer. The design, however, is limited in that only the structure of the non-covariate (i.e., non z) part of the model is explored relative to these effects. To understand to what extent the slopes depend on these factors, a model of the form

$$\theta_{ij} = \mu + \alpha_i + \beta_j + (\phi + \gamma_i + \epsilon_j + \gamma_{ij})k$$

must be analyzed. The first θ model therefore explores the dependence of the initial, stage one, or background failure rate on the two factors. The second representation of θ permits the testing of the dependence of the shocks to this failure rate as a function of these factors. Unfortunately the limited amount of data did not permit the exploration of this second representation.

The results of the nonlinear estimation process are summarized in Tables 5.5-1 and 5.5-2. For the model based on the first definition of k the tests conducted on the coefficients indicate that all ten of the fitted free parameters are significantly different from zero except α_1 and the conclusion on α_1 affects only the relative location of the three α parameters as they have sum zero. For the second definition of k, the tests indicate that all of the coefficients are significantly non zero. Thus for both models, the non k dependent part of the expansion of θ depends on both factors. That is, there is a significant programmer and a significant problem factor explaining the background failure rate.

TABLE 5.5-1
PROPORTIONAL HAZARDS MODEL PARAMETERS
BASED ON ORIGINAL ERROR RATE DATA

Coefficients	Variance	Standard Deviation	Significance ($\alpha = .05$)
Constant			
$\mu = -1.2602$	.002604	.05103	Sig.
Problem Levels			
$\alpha_1 = -.02389$	.006144	.07839	Not Sig.
$\alpha_2 = .5711$	.004629	.06803	Sig.
$\alpha_3 = -.5472$	--		
Programmer Levels			
$\beta_1 = .4158$	.002854	.05342	Sig.
$\beta_2 = -.4158$	--		

Slopes

$$\begin{pmatrix} \gamma_{A1} & \gamma_{A2} & \gamma_{A3} \\ \gamma_{B1} & \gamma_{B2} & \gamma_{B3} \end{pmatrix} = \begin{pmatrix} -1.1000 & -1.8562 & -.8002 \\ -1.0902 & -3.7898 & -1.0193 \end{pmatrix}$$

$$\text{Var}(\gamma) = \begin{pmatrix} .004406 & .001570 & .002754 \\ .0009986 & .007047 & .003288 \end{pmatrix}$$

$$\text{s.d.}(\gamma) = \begin{pmatrix} .06637 & .03963 & .05248 \\ .03160 & .08395 & .05734 \end{pmatrix}$$

First stage (background) error rate exponent (estimated)

$$(\log \lambda_{1J}) = (\mu + \alpha_1 + \beta_J) = \begin{pmatrix} -.8682 & .2732 & -1.3916 \\ -1.6999 & -.1.1049 & -2.2232 \end{pmatrix}$$

First stage failure rate (estimated)

$$(\lambda_{1J}) = \begin{pmatrix} .4197 & 1.3142* & .2487 \\ .1829 & .3312* & .1083 \end{pmatrix}$$

* Unconstrained model

TABLE 5.5-2
PROPORTIONAL HAZARDS MODEL PARAMETERS
BASED ON MODIFIED ORIGIN DATA

Coefficients	Variance	Standard Deviation	Significance ($\alpha = .05$)
Constant			
$\mu = -1.9394$	.003022	.05497	Sig.
Problem Levels			
$\alpha_1 = -.8740$	.007538	.08682	Sig.
$\alpha_2 = .5102$	.005507	.07421	Sig.
$\alpha_3 = .3638$	--	--	--
Programmer Levels			
$\beta_A = -.3026$	.003010	.05486	Sig.
$\beta_B = .3026$	--	--	--

Slopes

$$\begin{pmatrix} \gamma_{A1} & \gamma_{A2} & \gamma_{A3} \\ \gamma_{B1} & \gamma_{B2} & \gamma_{B3} \end{pmatrix} = \begin{pmatrix} -1.7196 & -2.0717 & -.5970 \\ -.9067 & -3.7781 & -1.4001 \end{pmatrix}$$

$$\text{Var}(\gamma) = \begin{pmatrix} .01097 & .003503 & .002663 \\ .001257 & .007115 & .003333 \end{pmatrix}$$

$$\text{s.d.}(\gamma) = \begin{pmatrix} .1047 & .05918 & .05161 \\ .0355 & .08435 & .05773 \end{pmatrix}$$

First stage (background) failure rate exponent (model)

$$(\log \lambda_{ij}) = (\mu + \alpha_i + \beta_j) = \begin{pmatrix} -3.1161 & -1.7319 & -1.8782 \\ -2.5109 & -1.1267 & -1.2730 \end{pmatrix}$$

First stage failure rate (model)

$$(\lambda_{ij}) = \begin{pmatrix} .04433 & .1770 & .1529 \\ .08120 & .3241 & .2800 \end{pmatrix}$$

A number of tests of equality were also conducted to determine if any of the pairwise γ's across problems for a given programmer or across programmers for a given problem were equal. Only isolated cases failed to reject so that some interaction between the two factors seems to exist. To what extent the main effects explain the slopes is not known.

Table 5.5-3 and Figures 5.5-1 and 5.5-2 compare the raw estimates of the p values and the ln p values to the predicted values based on the proportional hazards model with parameter estimates from Tables 5.5-1 and 5.5-2. In general the model does reasonably well in predicting log failure rate.

Since a methodology for evaluating the quality of the fit for the two models does not exist at this time, it is difficult to measure if the new definition of the initial state of the program increases the efficiency of the modeling. The variance of the residual not explained by the model can be compared, however, to the original data variance, based on the data in Figures 5.5-1 and 5.5-2, in order to obtain a pseudo R^2 value for comparison purposes. For model 1 the percentage reduction in variance due to the model is 80.7%. For model 2 the percentage reduction is 83.5%. Thus there is a slight but not appreciable advantage for this problem set in using the new definition of the initial state.

5.6 PROGRAM FEATURES AS PREDICTORS

Several physical features of the subject programs have been measured as possible predictors of the error failure rate. Table 5.6-1 gives these features for all the subject programs. In general these are in the form of counts of such features as program length, total branch modes including the statements GO TO, DO, IF and CALL and two of Halstead's [17] program measures; his length measure N and his total error predictor E. Both of Halstead's measures depend on counting the number of operators and the number of operands in the program, neither of which has been precisely defined for all cases. There is no particular rationale behind these choices of physical features for consideration except that "length" measures and "complexity" measures of a program seem reasonable as "first look" predictors. This set is cursory and is not intended to be exhaustive in any way.

In general, these program features are all poor predictors of both slope and first stage failure rate. Figure 5.6-1, however, demonstrates that the slopes of programmer B suggest a linearly increasing trend with the inverse of Halstead's length measure N. (This is also true of E because N and E are nearly linearly dependent in this range.) Thus, for this programmer, as Halstead's length increases the failure rate for fixed k increases.

The behavior of the programs written by programmer A are much less consistent with regard to this measure. A possible reason for this discrepancy might be that the programming time for this programmer was not as consistently controlled as for programmer B, particularly for program A1 due to outside factors. Program A1 seems to be an outlier with regard to every single physical measure and seems to have a much smaller failure rate than its length and complexity would indicate.

TABLE 5.5-3
ESTIMATES FOR ERROR PROBABILITY PER PROGRAM EXECUTION,
BASED ON THE PROPORTIONAL HAZARDS MODEL

PROGRAM	STAGE	EXPERIMENT BASED ESTIMATES	MODEL PREDICTIONS (ORIGINAL ORIGIN)	MODEL PREDICTIONS (MODIFIED ORIGIN)
A1	1	.9803	.4197	
	2	.1068	.05678	.04430
	3	.002602	.007689	.007936
	4	.002104	.001041	.001422
	5	.001176	.0001409	.0002546
	6	.0007659	.00001907	.00004561
B1	1	.1524	.1829	.0812
	4	.009117	.006947	.005348
	5	.003483	.002335	.002160
	6	.0005359	.007850	.0008721
A2	1	.8065	1.3142*	
	2	.2632	.2054	.1770
	3	.03759	.03209	.02230
	4	.01909	.005015	.002809
	5	.0001374	.0007836	.0003538
B2	1	.1449	.3312	.3241
	2	.02625	.007486	.007411
	3	.0001033	.0001692	.0001694
A3	1	.2488	.2487	.1529
	2	.05376	.1117	.08417
	3	.03524	.05109	.04633
	4	.02691	.02255	.02550
	5	.02302	.01012	.01404
B3	1	.3650	.1083	.2800
	2	.1037	.04864	.06904
	3	.01021	.02185	.01702
	4	.004681	.009816	.004198
	5	.002411	.004410	.001035

* Unconstrained Model

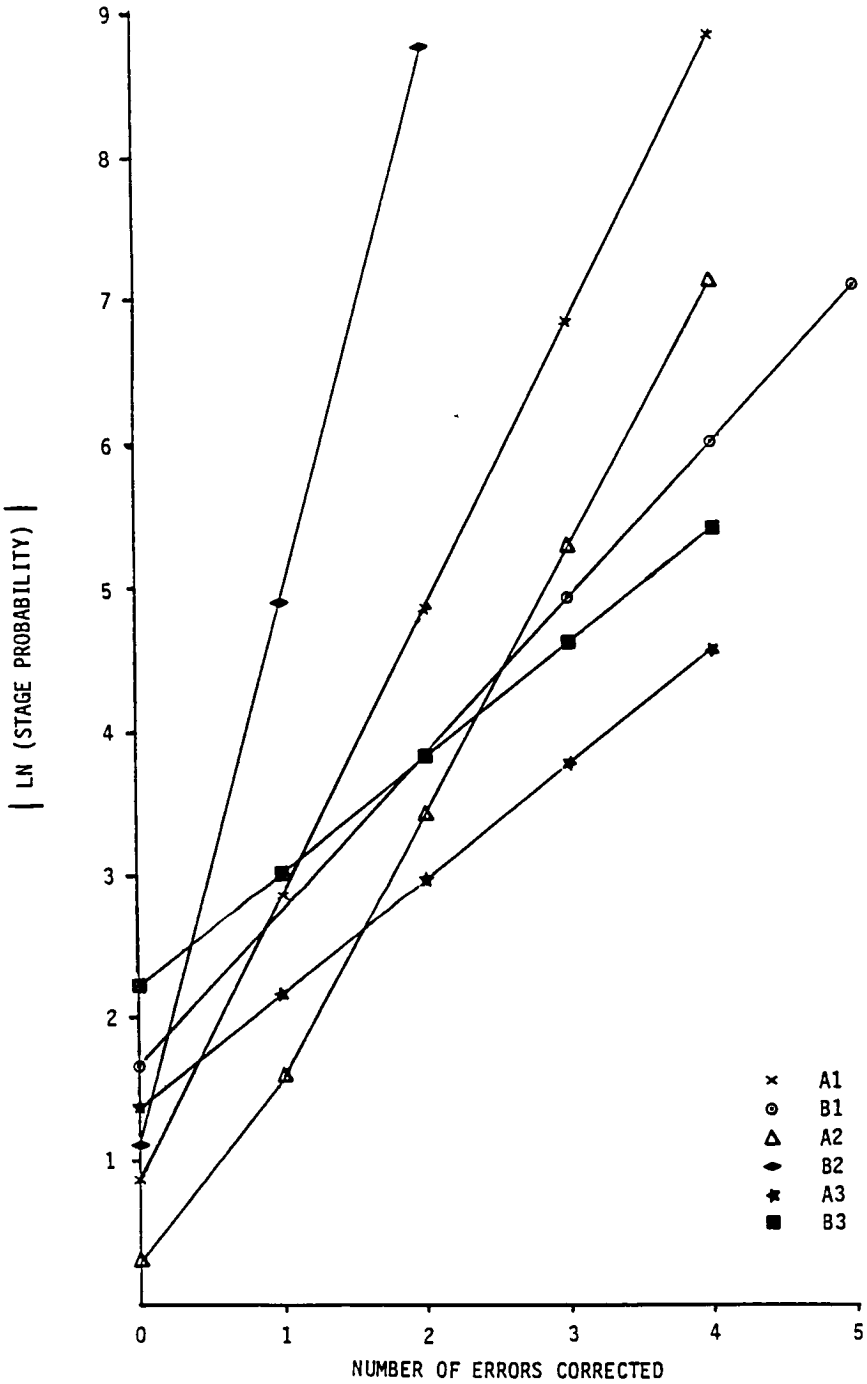


Figure 5.5-1. Predicted Error Rate from the Proportional Hazards Model, Using Original Data.

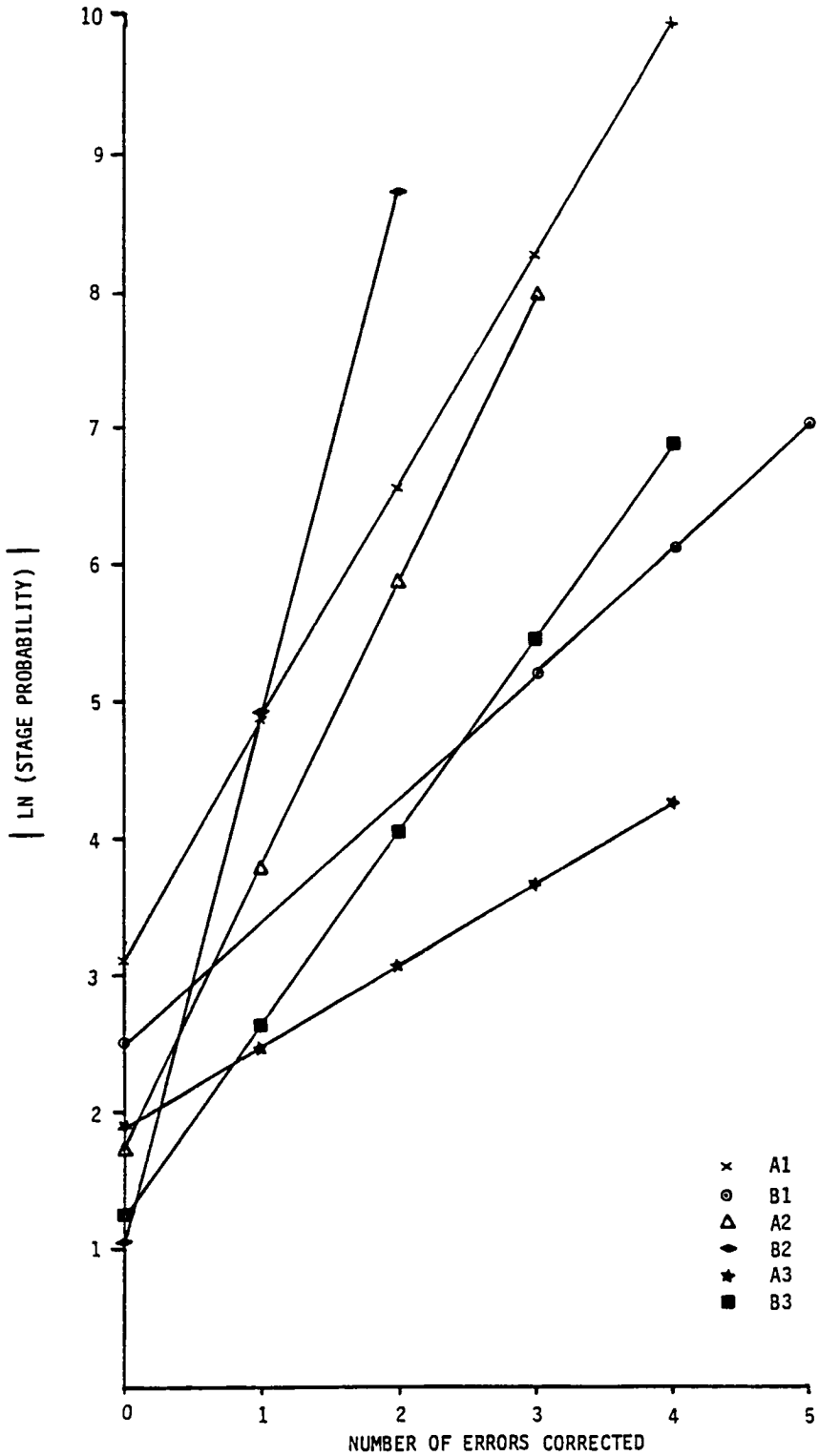


Figure 5.5-2. Predicted Error Rate from the Proportional Hazards Model, Using Modified-origin Data.

TABLE 5.6-1 SUBJECT PROGRAM MEASURES

Problem

Programmer	1	2	3
A	632	320	294
B	236	186	145

Length

Problem

Programmer	1	2	3
A	179	62	61
B	87	50	41

Branch Count

Programmer	1	2	3
A	232	150	126
B	107	109	79

Operators

Programmer	1	2	3
A	266	185	198
B	144	111	145

Operands

Programmer	1	2	3
A	2354	1453	1437
B	1239	902	1156

Halstead's Length N

Programmer	1	2	3
A	4.56	2.58	2.63
B	2.49	1.42	2.46

Halstead's Error Prediction E

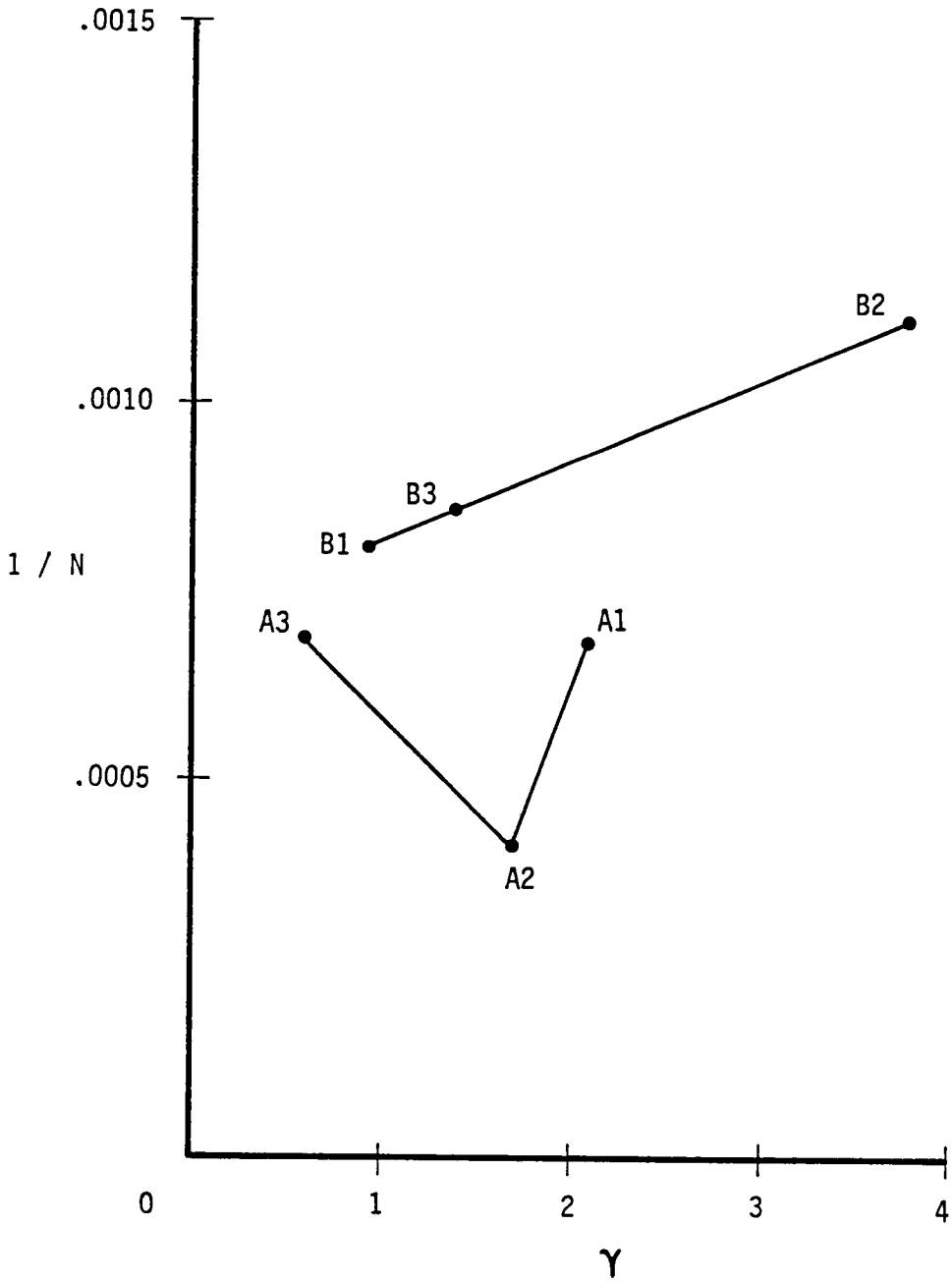


Figure 5.6-1. Inverse of Halstead's Length vs. Predicted Slope (Modified Data), Proportional Hazards Model.

5.7 RIPPLE EFFECT

In only one program did the correction of an error introduce additional errors. Since so little data was available with which to model this effect, no attempt was made to include it as a special case in the proportional hazards model. It suffices to point out that the model is sufficiently general to include shocks that increase as well as decrease the failure rate.

6.0 CONCLUSIONS

The results of this experiment have proven useful in exploring the foundations of the probabilistic process of detecting software errors. The data has been used to statistically verify that interfailure time is exponentially distributed and to prove that errors can occur with widely different failure rates. This fact invalidates many of the more popular software reliability models now in use. It was also observed that log failure rate was nearly linear as a function of the number of errors corrected.

It was demonstrated that Cox's proportional hazards life model has all of the observed characteristics of the data noted above and is proposed as a new model for predicting software reliability. The model specifies a log linear hazard function depending on a covariate representing the number of errors corrected. Maximum likelihood estimates for the unknown parameters of this model were obtained for all six subject programs using nonlinear optimization techniques. Tests on these estimates indicate that there are strong programmer and problem effects in the background failure rate. Results also show that over 80% of the observed variation in the logarithm of the failure rate data can be explained by the proportional hazards function model. A tentative physical predictor was proposed based on Halstead's information criterion N that might prove useful in forecasting model parameters.

Forecasting software reliability based on physical program features is a subject in its infancy. Probabilistic models with parameters that require statistical estimation from operational experience on a program do, however, offer forecasting alternatives. If operational usage data can be simulated, then repetitive run sampling designs provide a rich statistical base for estimating these parameters. Repetitive run sampling is no different from the standard method of recording software error data in a single run, with regard to the number of different errors detected. Both methods will uncover the same errors on the average in the same number of program executions. The differences occur in the amount and kind of interim information, on the frequency of error occurrence and the effects of conditioning, recorded during testing. These potential uses of the additional information gained during repetitive sampling can only be conjectured, but based on the experiences of this study it appears very promising.

This is the first software reliability model to be based on the results of a carefully designed and controlled software experiment. These results indicate that software error detection has many of the attributes of a predictable structure. The process of completely understanding the nature of the problem, however, has just started and much additional research is necessary to substantiate these results on more complicated problems written by more experienced personnel under the same controlled conditions.

REFERENCES

1. Jelinski, F. and Moranda, P. B., "Software Reliability Research," Statistical Computer Performance Evaluation, W. Freiberger, Ed., New York: Academic, 1972.

2. Shooman, M. L., "Probabilistic Models for Software Reliability Prediction," Probabilistic Models for Software, W. Freiberger, Ed., New York: Academic, 1972.

3. Schick, G. J. and Wolverton, R. W., "An Analysis of Competing Software Reliability Models," IEEE Transactions on Software Engineering, Vol. SE-4, No. 2, March 1978.

4. Littlewood, B. and Verrall, J. L., "A Bayesian Reliability Growth Model for Computer Software," 1973 IEEE Symposium on Computer Software Reliability, New York, 1973.

5. Littlewood, B., "What Makes A Reliable Program - Few Bugs, or a Small Failure Rate?," AFIPS Conference Proceedings, Vol. 49, 1980.

6. Brown, J. R. and Lipow, M. "Testing for Software Reliability," Proceedings, 1975 International Conference on Reliable Software, IEEE Cat. No. 75-CH0940-7CSR, 1975.

7. Migneault, G. E., " Emulation Applied to Reliability Analysis of Reconfigurable, Highly Reliable, Fault-Tolerant Computing Systems," AGARD Conference Proceeding, No. 261, 1980.

8. Fosdick, Lloyd D. and Osterweil, L. J., "DAVE - A Fortran Program Analysis System," Computer Science and Statistics, 8th Annual Symposium on the Interface, 1975.

9. Stucki, L. G. and Foshee, G. L., "New Assertion Concepts for Self Metric Software Validation," Proceedings, 1975 International Conference on Reliable Software, IEEE Cat. No. 75-CH0940-7CSR, 1975.

10. Brown, J. R. and Buchanan, H. N., "The Quantitative Measurement of Software Safety and Reliability," TRW SDP 1776, TRW Systems Group, Redondo Beach, California, 1973.

11. Finkelstein, J. M. and Schafer, R. E., "Improved Goodness-of-Fit Tests," Biometrika, 58:3, pp. 641-645, 1971.

12. Lilliefors, H. W., "On the Kolmogoroff-Smirnov Test for the Exponential Distribution With Mean Unknown," Journal of the American Statistical Association, pp. 387-389, March 1980.

13. Mann, N. R., Schafer, R. E., and Singpurwalla, Nozer D., Methods for Statistical Analysis of Reliability and Life Data, Wiley: New York, 1974.

14. Barlow, Richard E. and Proschan, Frank, Mathematical Theory of Reliability, Wiley: New York, 1965.

15. Kalbfleisch, J. D. and Prentice, R. L., The Statistical Analysis of Failure Time Data, Wiley: New York, 1980.

16. Cox, D. R., "Regression Models and Life Tables," Journal of the Royal Statistical Society, Series B, Vol. 34, pp. 187-220, 1972.

17. Halstead, M. A., Elements of Software Engineering, Elsenier: New York, 1977.

APPENDIX A: SOFTWARE ERROR CATEGORIES

From Brown and Buchanan [10]

A000		COMPUTATIONAL ERRORS
	A100	Incorrect operand in equation
	A200	Incorrect use of parenthesis
	A300	Sign convention error
	A400	Units or data conversion error
	A500	Computation produces an over/under flow
	A600	Incorrect/inaccurate equation used
	A700	Precision loss due to mixed mode
	A800	Missing computation
	A900	Rounding or truncation error
B000		LOGIC ERRORS
	B100	Incorrect operand in logical expression
	B200	Logic activities out of sequence
	B300	Wrong variable being checked
	B400	Missing logic or condition tests
	B500	Too many/few statements in loop
	B600	Loop iterated incorrect number of times (including endless loop)
	B700	Duplicate logic
C000		DATA INPUT ERRORS
	C100	Invalid input read from correct data file
	C200	Input read from incorrect data file
	C300	Incorrect input format
	C400	Incorrect format statement referenced
	C500	End of file encountered prematurely
	C600	End of file missing
D000		DATA HANDLING ERRORS
	D050	Data file not rewound before reading
	D100	Data initialization not done
	D200	Data initialization done improperly
	D300	Variable used as a flag or index not set properly
	D400	Variable referred to by the wrong name
	D500	Bit manipulation done incorrectly
	D600	Incorrect variable type
	D700	Data packing/unpacking error
	D800	Sort error
	D900	Subscripting error

E000		DATA OUTPUT ERRORS
	E100	Data written on wrong file
	E200	Data written according to the wrong format statement
	E300	Data written in wrong format
	E400	Data written with wrong carriage control
	E500	Incomplete or missing output
	E600	Output field size too small
	E700	Line count or page eject problem
	E800	Output garbled or misleading
F000		INTERFACE ERRORS
	F100	Wrong subroutine called
	F200	Call to subroutine not made or made in wrong place
	F300	Subroutine arguments not consistent in type, units, order, etc.
	F400	Subroutine called is nonexistent
	F500	Software/data base interface error
	F600	Software/user interface error
	F700	Software/software interface error
G000		DATA DEFINITION ERRORS
	G100	Data not properly defined/dimensioned
	G200	Data referenced out of bounds
	G300	Data being referenced at incorrect location
	G400	Data pointers not incremented properly
H000		DATA BASE ERRORS
	H100	Data not initialized in data base
	H200	Data initialized to incorrect value
	H300	Data units are incorrect
I000		OPERATION ERRORS
	I100	Operating system error (vendor supplied)
	I200	Hardware error
	I300	Operator error
	I400	Test execution error
	I500	User misunderstanding/error
	I600	Configuration control error

J000		OTHER
	J100	Time limit exceeded
	J200	Core storage limit exceeded
	J300	Output line limit exceeded
	J400	Compilation error
	J500	Code or design inefficient/not necessary
	J600	User/programmer requested enhancement
	J700	Design nonresponsive to requirements
	J800	Code delivery or redelivery
	J900	Software not compatible with project standards
K000		DOCUMENTATION ERRORS
	K100	User manual
	K200	Interface specification
	K300	Design specification
	K400	Requirements specification
	K500	Test documentation
X0000		PROBLEM REPORT REJECTION
	X0001	No problem
	X0002	Void/withdrawn
	X0003	Out of scope - not part of approved design
	X0004	Duplicates another problem report
	X0005	Deferred

APPENDIX B: PROBLEM #1 SPECIFICATIONS

From Brown and Buchanan [10]

1.0 LAUNCH INTERCEPTOR CONDITIONS (LIC)

Conditions were specified in such a way that the resulting program would be similar to a Site Defense program attempting to correlate radar tracking returns. Nineteen parameters were required as input to precisely specify these conditions. The Launch Interceptor Conditions (LIC) were defined as follows:

1) Any two consecutive data points are a distance greater than the length, ℓ, apart.

2) Any three consecutive data points cannot all be contained within or on a circle of radius r.

3) Any three consecutive data points form an angle, a, where $a < (\pi - \epsilon_2)$ or $a > (\pi + \epsilon_2)$. Being measured here is the angle a's deviation from 180 degrees. The second of the three consecutive points is always at the vertex of the angle.

4) Any three consecutive data points form a triangle with area greater than A. The three points are at the triangle's vertices.

5) Any M consecutive data points lie in more than Q quadrants. Where there is ambiguity as to which quadrant contains a given point, priority of decision will be by quadrant number, i.e., I, II, III, IV. For example, the data point (0,0) is in quadrant I. Also, the point (-1,0) is in quadrant II. The point (0,-1) is in quadrant III.

6) For any two consecutive data points, P_1 and P_2, the difference of their abscissas is negative, i.e., $(X_2 - X_1) < 0$.

7) At least one of any N consecutive data points lies a distance greater than ϵ_1 from the line joining the first and last of these points.

8) Any two data points (with n_1 consecutive intervening points) are a distance greater than the length, ℓ, apart.

9) Any three data points (with n_2 and m_2 consecutive intervening points, respectively) cannot be contained within or on a circle of radius r.

10) Any three data points (with n_3 and m_3 consecutive intervening points, respectively) form an angle, α, where $\alpha < (\pi - \epsilon_2)$ or $\alpha > (\pi + \epsilon_2)$. Being measured here is the angle α's deviation from 180 degrees. Of the above first mentioned three data points, the second is always at the vertex of the angle.

11) Any three data points (with n_4 and m_4 consecutive intervening points, respectively) form a triangle with area greater than A. The above first mentioned three data points are at the triangle's vertices.

12) For any two data points, P_1 and P_2 (with n_6 consecutive intervening points) the difference of their abscissas is negative, i.e., $(X_2 - X_1)<0$.

13) Any two data points, with n_1 consecutive intervening points, are a distance greater than the length, ℓ, apart. Also, any two data points (which can be the same or different from the above first mentioned two data points), with n_1 consecutive intervening points, are a distance less than the length, L, apart.

14) Any three data points, with n_2 and m_2 consecutive intervening points, respectively, cannot be contained within or on a circle of radius r. Also, any three data points (which can be the same or different from the above first mentioned three data points), with n_2 and m_2 consecutive intervening points, respectively, can be contained in or on a circle of radius R.

15) Any three data points, with n_4 and m_4 consecutive intervening points, respectively, form a triangle with area greater than A. The above first mentioned three data points are at the triangle's vertices. Also, any three data points (which can be the same or different from the above first mentioned three data points), with n_4 and m_4 consecutive intervening points, respectively, form a triangle with area less than E. The above second mentioned three data points are at the (second) triangle's vertices.

2.0 PROBLEM LOGIC

1) Information was supplied indicating the logical connectors among all the LIC, as defined in Section 1.0. The format was a symmetrical square matrix where zero indicated NOT used, one indicated the OR connector between two conditions and two indicated the AND connector. The matrix was identified as the Logical Connector Matrix (LCM).

2) Part of the output data was a column matrix with resultant information as to whether or not the LIC were met, i.e., for each condition, zero meant the condition was not met and one meant it was met. The identification for this matrix was Conditions Met Matrix (CMM).

3) Preliminary unlocking information was generated. By definition, these were criteria which determined whether or not interceptors would be launched. These data were determined by the interaction of the LCM and CMM matrices to form the Preliminary Unlocking Matrix (PUM). Definitions of the matrix elements indicate how the two matrices interact to form PUM. The diagonal elements of PUM were input according to the desired or required unlocking sequence, i.e., a one indicated that the corresponding LIC was to be considered as a factor in signaling interceptor launch and a zero meant that it was not a factor. Non-diagonal elements were determined by the LCM operating as a Boolean operator, as defined in Section 2.0.1, on the operand CMM.

4) The Final Unlocking Matrix (FUM) was generated by having the PUM diagonal operate on non-diagonal elements. An element in the FUM was one (1) if:

The corresponding PUM diagonal element was zero (0), indicating no interest in the associated LIC; or

The corresponding PUM diagonal element was one (1) and all other elements in that diagonal element's row were one (1).

An element in the FUM represented the final conclusion with respect to its corresponding LIC.

5) In order to launch an interceptor, all elements in FUM had to be equal to one. In this case, the message "NOW" was generated and output to the printer, together with a listing of all input data values. The information from all matrices was printed. The output was in matrix format for ease of interpretation.

3.0 DATA INFORMATION

1) Pairs of values for the rectangular coordinates (x, y) represented data points.

2) An input data set contained a maximum of 100 ordered data points.

3) P = number of data points in a data set.

4) The input data constants, as defined in Section 1.0 were specified for each input data set.

5) Restrictions on the input parameters were as follows:

$P > 2,\ \ell \geq 0,\ r \geq 0,\ 0 \leq \epsilon_2 < \pi,\qquad A \geq 0,\ M \leq P,\ 1 \leq Q \leq 3,$

$\epsilon_1 \geq 0,\ L \geq 0,\ R \geq 0,\ E \geq 0.$

6) The Logical Connector Matrix (LCM) element values were given as input.

7) The Preliminary Unlocking Matrix (PUM) diagonal element values were given.

For the actual data values, see the example matrices in the following section.

4.0 EXAMPLE MATRICES

The following matrices are a model of the problem logic, as defined in Section 2.0.

Logical Connnector Matrix (LCM) (Input)

*	1	2	3	4	5 . . . 15
1	2	2	1	2	0 . . . 0
2	2	2	1	1	0 . . . 0
3	1	1	2	1	0 . . . 0
4	2	1	1	2	0 . . . 0
5	0	0	0	0	0 . . . 0
.	.	.	.	.	. . . .
.	.	.	.	.	. . . .
.	.	.	.	.	. . . .
15	0	0	0	0	0 . . . 0

*, Launch Interceptor Conditions (LIC)

Since we have zeros beyond the fourth LIC, the 5th through the 15th LIC are not to be considered in this example.

Definition - L_{ij} is the ij^{th} element in the LCM.

Conditions Met Matrix (CMM) (Output)

Condition	Value
1	0
2	1
3	1
4	0
5	0
.	.
.	.
.	.
15	0

Definition: C_i is the i^{th} element in the CMM.

The C_i are computed output, but in order to illustrate this example, we are arbitrarily setting these elements in the CMM.

Preliminary Unlocking Matrix (PUM)

(Output, non-diagonal elements)
(Input, diagonal elements)

LIC	1	2	3	4	5 . . . 15
1	1	0	1	0	1 . . . 1
2	0	0	1	1	1 . . . 1
3	1	1	1	1	1 . . . 1
4	0	1	1	0	1 . . . 1
5	1	1	1	1	0 . . . 1
.	.	.	.	.	. . .
.	.	.	.	.	. . .
.	.	.	.	.	. . .
15	.	.	.	.	. . 0

Furthermore, defining the ij^{th} element in the PUM as P_{ij}, we have the following: $P_{11} = P_{33} = 1$ and all other P_{ii} (i.e., the diagonal elements) are zero. This means that only the first and third LIC are required in the unlocking sequence. Note that these are input values.

$P_{12} = 0$ since, $L_{12} = 2$, signifying the AND condition for C_1 and C_2 which are zero and one, respectively, i.e., 01 = 0.

$P_{13} = 1$ since, $L_{13} = 1$, signifyng the OR condition for C_1 and C_3 which are zero and one, respectively, i.e., 0 + 1 = 1.

$P_{14} = 0$ since, $L_{14} = 2$, signifying the AND condition for C_1 and C_4 which are both zero, i.e., 00 = 0.

$P_{15} = 1$ since $L_{15} = 0$, signifying the Not Used condition for C_1 and C_5. The above examples show how to generate the P_{ij} values.

Final Unlocking Matrix (FUM) (Output)

LIC	VALUE
1	0
2	1
3	1
4	1
5	1
.	.
.	.
.	.
15	1

Definition: F_1 is the i^{th} element in the FUM.

$F_1 = 0$ since $P_{11} = 1$ and $P_{12} = P_{14} = 0$, i.e., the diagonal value is one and there is at least one zero element in the first row of PUM.

$F_2 = 1$, since $P_{22} = 0$.

$F_3 = 1$, since $P_{33} = 1$ and $P_{31} = P_{32} = P_{34} = P_{35} = \ldots = P_{3,15} = 1$.

$F_4 = 1$, since $P_{44} = 0$.

$F_5 = F_6 = \ldots = F_{15} = 1$, since, $P_{55} = P_{66} = \ldots = P_{15,15} = 0$, respectively.

Since there is a zero element in FUM, ($F_1 = 0$), the launch interceptor condition is not met.

5.0 SUPPLEMENTARY INFORMATION

1. The program will be written in FORTRAN on the BITS system.

2. No double precision or complex variables are required.

3. Your program will be a subroutine.

4. Assume the inputs are in labeled common, i.e., COMMON/INPUTS/ X(100), Y(100), EL, ... using the order in Section 6.0. You are free to use your own variable names, however.

5. Outputs will be in labeled common, i.e., COMMON/OUTPTS/CMM(15), ... using the order in Section 7.0. Again you are free to use your own variable names.

6. Use the IFOUT flag to control printing. Code the output statements, but branch around them if IFOUT = 1.

7. When the first and last points of N consecutive data points are identical, then the calculated distance to compare with ϵ (LIC #7), will be that distance from the coincident point to all others of the N consecutive points.

6.0 INPUTS

1. Data Points (x_i, y_i) $i=1, \ldots, P$ $2 \leq P \leq 100$

   accurate to one decimal place real

2. Nineteen Parameters

1.	l	real
2.	r	"
3.	ϵ_2	"
4.	A	"
5.	M	integer
6.	Q	"
7.	ϵ_1	real
8.	n_1	integer
9.	n_2	"
10.	m_2	"
11.	n_3	"
12.	m_3	"
13.	n_4	"
14.	m_4	"
15.	n_6	"
16.	L	real
17.	R	real
18.	E	real
19.	N	integer

3.	LCM Array	$LCM_{i,j}$	i=1,. . .,15 j=1,. . .,15	integer
4.	PUM array Diagonal Terms	$PUM_{i,i}$	i=1,. . .,15	integer
5.	P - number of data points			integer
6.	IFOUT - Controlling output, i.e.,		= 0 Program prints output = 1 Program prints no output integer	

7.0 OUTPUTS

1.	Conditions Met Matrix	CMM_i,	i=1,-,15	integer
2.	Final Unlocking Matrix	FUM_i,	i=1,-,15	integer

3. "LAUNCH" or "NO LAUNCH"

(Use a logical variable which is true for launch and false for no launch).

APPENDIX C: PROBLEM #1 TEST CASES

From Brown and Buchanan [10]

1.0 TEST CASE 1

- Input
  - 100 (X,Y) points defined by $(X_i,Y_i) = (2i - 2, i - 1)$ for $i = 1,...,100$
  - LIC parameters = $\ell = 2.3$, $r = 2.3$, $\epsilon_2 = 0.2$, $A = 0.3$, $M = 4$, $Q = 1$, $\epsilon_1 = 0.1$, all n_i and m_i equal to 0, $L = 2$, $R = 2.2$, $E = 0$, $N = 5$.
  - LCM:

LIC	1	2	3	4	5	...15
1	2	2	1	2	0	... 0
2	2	2	1	1	0	... 0
3	1	1	2	1	0	... 0
4	2	1	1	2	0	... 0
5	0	0	0	0	0	... 0
.	.	.	.	.	.	... 0
.	.	.	.	.	.	... .
.	.	.	.	.	.	... .
15	0	0	0	0	0	... 0

  - PUM diagnoal: $P_{11} = P_{33} = 1$; all other P_{ii} equal to zero.
- Output
  - PUM:

LIC	1	2	3	4	5	...15
1	1	0	0	0	1	... 1
2	0	0	0	0	1	... 1
3	0	0	1	0	1	... 1
4	0	0	0	0	1	... 1
5	1	1	1	1	0	... 1
.	.	.	.	.	.	... .
.	.	.	.	.	.	... .
.	.	.	.	.	.	... .
15	1	1	1	1	1	... 0

- CMM: all 0's
- FUM: 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1

- Final Conclusion: No Launch

2.0 TEST CASE 2

- Input
  - 95 (X,Y) points defined by $(X_i, Y_i) = (2i - 2, i - 1)$ for $i = 1,...,96$ except $i \neq 5$
  - LIC parameters: same as Test Case 1.
  - LCM:

LIC	1	2	3	4	5	...15
1	2	1	2	2	0	... 0
2	1	2	1	1	0	... 0
3	2	1	2	1	0	... 0
4	2	1	1	2	0	... 0
5	0	0	0	0	0	... 0
.	.	.	.	.	.	... .
.	.	.	.	.	.	... .
.	.	.	.	.	.	... .
15	0	0	0	0	0	... 0

  - PUM diagonal: $P_{11} = P_{22} = 1$ and all other $P_{ii} = 0$

- Output
  - PUM:

LIC	1	2	3	4	5	...15
1	1	1	0	0	1	... 1
2	1	1	1	1	1	... 1
3	0	1	0	0	1	... 1
4	0	1	0	0	1	... 1
5	1	1	1	1	0	... 1
.	.	.	.	.	.	... .
.	.	.	.	.	.	... .
.	.	.	.	.	.	... .
15	1	1	1	1	1	... 0

- CMM: 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0
- FUM: 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
- Final Conclusion: No Launch

3.0 TEST CASE 3

- Input
  - (X,Y) points same as Test Case 2
  - LIC parameters: same as Test Case 1
  - LCM:

LIC	1	2	3	4	5	...15
1	2	2	1	1	0	... 0
2	2	2	1	1	0	... 0
3	1	1	2	1	0	... 0
4	1	1	1	2	0	... 0
5	0	0	0	0	0	... 0
.	.	.	.	.	.	... .
.	.	.	.	.	.	... .
.	.	.	.	.	.	... .
15	0	0	0	0	0	... 0

  - PUM diagonal: same as Test Case 1
- Output
  - PUM:

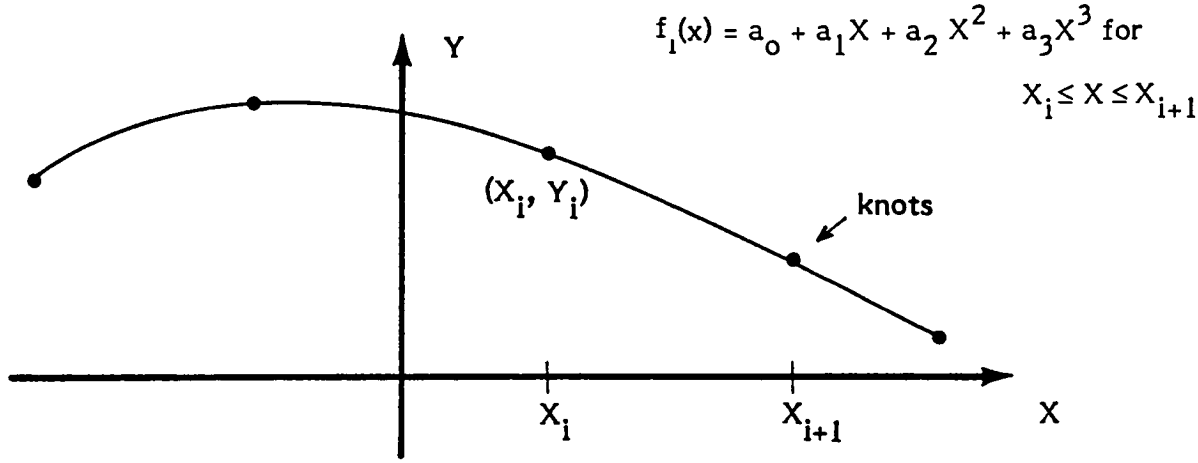
LIC	1	2	3	4	5	...15
1	1	1	1	1	1	... 1
2	1	1	1	1	1	... 1
3	1	1	0	0	1	... 1
4	1	1	0	0	1	... 1
5	1	1	1	1	0	... 1
.	.	.	.	.	.	... .
.	.	.	.	.	.	... .
.	.	.	.	.	.	... .
15	1	1	1	1	1	... 0

- CMM: 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0
- FUM: all 1's
- Final Conclusion: Launch

APPENDIX D: PROBLEM #2 SPECIFICATIONS

1.0 GENERAL

Consider the following diagram:



A set of piecewise cubic polynomials passing through predefined (x, y) coordinates, or knots, is a cubic spline if the functional value and the first and second derivatives of two adjoining cubic polynomials are continuous at the knot where they join.

For n knots, there are n-1 cubics. Since each cubic has 4 unknowns, there are 4n-4 unknowns in this cubic spline. The 3 continuity conditions at the n-2 inner points give 3n-6 conditions. There are n other conditions because the spline must pass through the knots. In addition, we need two more conditions to have 4n-4 conditions and 4n-4 unknowns. These additional conditions will be to specify the second derivative at each endpoint.

Note: An alternate form of the cubic over an interval $[X_i, X_{i+1}]$ is

$$f_i(X) = y_i + a(X-X_i) + b(X-X_i)^2 + C(X-X_i)^3.$$

2.0 DESCRIPTION OF SUBROUTINES

Four subroutines are required for Problem #2:

1. SPLINE to calculate coefficients of a cubic spline passing through given knots. Use the first cubic representation.

2. ADJUST to calculate coefficients of the alternate form of the cubic (see above).

3. SINTRP to interpolate the functional value and the first two derivatives at an input x_o using the coefficients from ADJUST.

4. SINTEG to integrate from X_1 to X_2 the spline generated using coefficients from ADJUST.

Numerical results must be within 1.0% (relative error) of comparable results from library routines.

2.1 SUBROUTINE SPLINE (X, Y, N, SDL, SDR, COEFF, IERR)

INPUT N, # of knots, $2 \leq N \leq 6$
X(I), Y(I), knots (coordinates), in any order
SDL, second derivative at left-most knot
SDR, second derivative at right-most knot

OUTPUT $\{X(I), Y(I)\}$, in ascending X order.
((COEFF(I,J), I=1, 4), J=1, N-1)

For a given interval J, $[X_J, X_{J+1}]$

COEFF(1,J) = a_o

COEFF(2,J) = a_1

COEFF(3,J) = a_2

COEFF(4,J) = a_3

where $f(X) = a_o + a_1X + a_2X^2 + a_3X^3$
is the cubic in that interval.

IERR = 0 for normal return
= 2 for two or more X coordinates identical
= 8 for a singular matrix

2.2 SUBROUTINE ADJUST (X, Y, N, COEFF, COEFAD)

INPUT N, # of knots, $2 \leq N \leq 6$
X, X-coordinates of knots in ascending order
Y, Y-coordinates of knots corresponding to the x-coordinates
COEFF, spline coefficients resulting from SUBROUTINE SPLINE

OUTPUT ((COEFAD(I, J), I=1, 3), J=1, N-1)

For a given interval J, $\left[X_J, X_{J+1}\right]$

COEFAD (1, J) = a

COEFAD (2, J) = b

COEFAD (3, J) = c

where $f_J(X) = Y_J + a\,(X-X_J) + b(X-X_J)^2 + c(X-X_J)^3$ is the cubic in that interval.

2.3 SUBROUTINE SINTRP (N, X, Y, COEFAD, XO, YO, YOP, YOPP, IERR)

INPUT N, # of knots, $2 \leq N \leq 6$
X, X-coordinates of knots in ascending order
Y, Y-coordinates of knots corresponding to the X-coordinates
COEFAD, spline coefficients resulting from SUBROUTINE ADJUST
XO, arbitrary X coordinate

OUTPUT YO, interpolated value.
YOP, first derivative at XO.
YOPP, second derivative at XO.
IERR = 0 for normal return
= 1 if XO < X(1)
= 3 if XO > X(N)

2.4 SUBROUTINE SINTEG (N, X, Y, COEFAD, X1, X2, S, IERR)

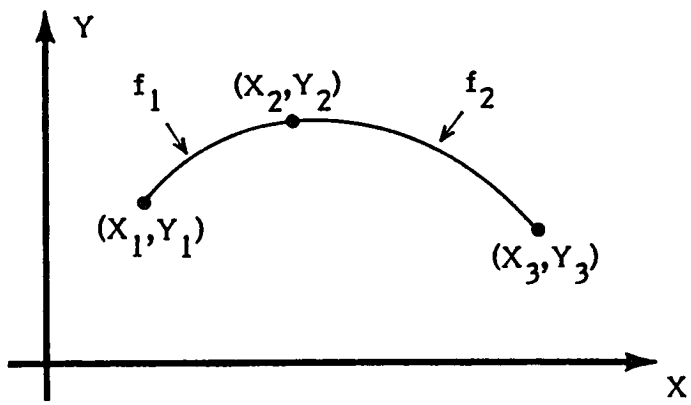
INPUT N, # of knots, $2 \leq N \leq 6$
X, X-coordinates of knots in ascending order
Y, Y-coordinates of knots corresponding to X-coordinates
COEFAD, spline coefficients resulting from SUBROUTINE ADJUST
X1, X2, endpoints for integration

OUTPUT S, integral of the spline function from X1 to X2. Return 0. if the endpoints of the spline don't span [X1, X2] or [X2, X1].

IERR = 0 for normal return
= 1 if endpoints of spline don't span [X1, X2] or [X2, X1].

Note: If IERR $\neq$ 0 after calling SPLINE, then ADJUST, SINTRP and SINTEG will not be called.

3.0 SUPPLEMENTARY INFORMATION



$$f_1 = a_0 + a_1X + a_2X^2 + a_3X^3 \quad X_1 \leq X \leq X_2$$

$$f_2 = b_0 + b_1X + b_2X^2 + b_3X^3 \quad X_2 \leq X \leq X_3$$

Solve for a in the matrix equation Ca = d

$$
\begin{array}{c}
 \\
\textcircled{1} \\
\textcircled{2} \\
\textcircled{3} \\
\textcircled{4} \\
\textcircled{5} \\
\textcircled{6} \\
\textcircled{7} \\
\textcircled{8}
\end{array}
\overset{\text{C}}{\begin{pmatrix}
0 & 0 & 2 & 6X_1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 2 & 6X_3 \\
1 & X_1 & X_1^2 & X_1^3 & 0 & 0 & 0 & 0 \\
1 & X_2 & X_2^2 & X_2^3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & X_2 & X_2^2 & X_2^3 \\
0 & 0 & 0 & 0 & 1 & X_3 & X_3^2 & X_3^3 \\
0 & 1 & 2X_2 & 3X_2^2 & 0 & -1 & -2X_2 & -3X_2^2 \\
0 & 0 & 2 & 6X_2 & 0 & 0 & -2 & -6X_2
\end{pmatrix}}
\overset{\text{a}}{\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}}
=
\overset{\text{d}}{\begin{pmatrix} S_0 \\ S_1 \\ Y_1 \\ Y_2 \\ Y_2 \\ Y_3 \\ 0 \\ 0 \end{pmatrix}}
$$

Explanation of equations:

① 2nd derivative at left-most endpoint given, S_0

$f'_1 = a_1 + 2a_2X + 3a_3X^2$

$f''_1 = 2a_2 + 6a_3X$

$f''_1(X_1) = S_0 \Rightarrow 2a_2 + 6a_3X_1 = S_0$

② 2nd derivative at right-most endpoint is given, S_1

$f''_2(X) = 2b_2 + 6b_3X$

$f'_2(X_3) = S_1 \Rightarrow 2b_2 + 6b_3X_3 = S_1$

③ f_1 passes through (X_1, Y_1)

$f_1(X_1) = a_0 + a_1X_1 + a_2X_1^2 + a_3X_1^3 = Y_1$

④ f_1 passes through (X_1, Y_1)

⑤ f_2 passes through (X_2, Y_2)

⑥ f_2 passes through (X_3, Y_3)

⑦ $f'_1(X_2) = f'_2(X_2)$ (first derivative continuous at (X_2, Y_2))

$$a_1 + 2a_2X_2 + 3a_3X_2^2 = b_1 + 2b_2X_2 + 3b_3X_2^2$$

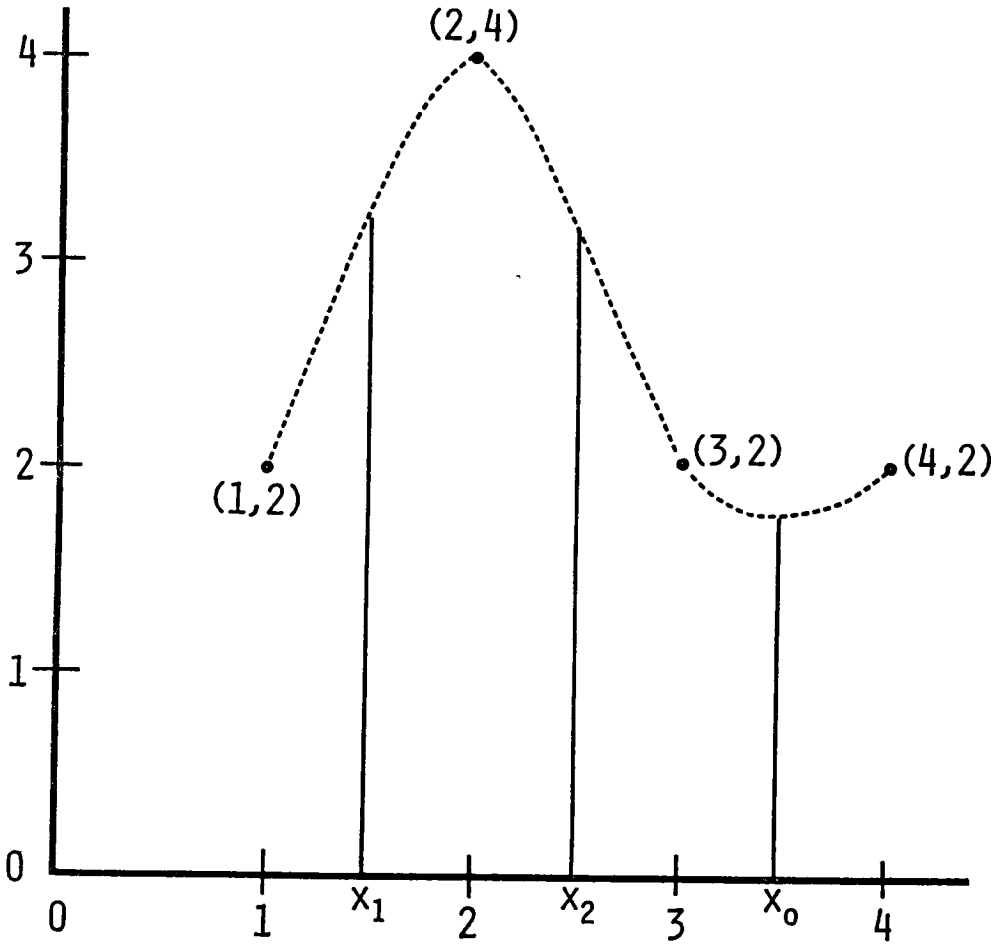
$$a_1 + 2a_2X_2 + 3a_3X_2^2 - b_1 - 2b_2X_2 - 3b_3X_2^2 = 0$$

⑧ $f''_1(X_2) = f''_2(X_2)$ (second derivative continuous at (X_2, Y_2))

$$2a_2 + 6a_3X_2 = 2b_2 + 6b_3X_2$$

$$2a_2 + 6a_3X_2 - 2b_2 - 6b_3X_2 = 0$$

APPENDIX E: PROBLEM #2 TEST CASE



INPUT N=4

I	X(I)	Y(I)
1	1.	2.
2	2.	4.
3	3.	2.
4	4.	2.

SDL=0.

SDR=0.

X0 = 3.5

X1 = 1.5

X2 = 2.5

OUTPUT

SPLINE OUTPUT --

X(I), Y(I) 1.0000 2.0000 2.0000 4.0000 3.0000 2.0000
4.0000 2.0000

COEFF, SPLINE 1 .000000 -.400000 3.600000 -1.200000

COEFF, SPLINE 2 -25.600000 38.000000 -15.600000 2.000000

COEFF, SPLINE 3 50.000000 -37.600000 9.600000 -.800000

IERR (SPLINE) = 0

ADJUST OUTPUT --

COEFF, SPLINE 1 3.200000 .000000 -1.200000

COEFF, SPLINE 2 -.400000 -3.600000 2.000000

COEFF, SPLINE 3 -1.600000 2.400000 -.800000

SINTRP+SINTEG -- Y0 = 1.700000 Y0P = .200000 Y0PP = 2.400000 IERR(SINTRP) = 0

S = 3.750000 IERR(SINTEG) = 0

APPENDIX F: PROBLEM #3 SPECIFICATIONS

1.0 GENERAL

Given a spherical earth, any point on its surface can be described by two angles (spherical polar coordinates). These angles are specified by defining a pair of reference planes. The first plane is called the "equator" plane, and this plane divides the earth into two hemispheres: the northern hemisphere and the southern hemisphere. The second plane, the "Greenwich" plane, is normal to the equator plane (it contains the polar axis) and also divides the earth into two hemispheres, east and west. One of the angles mentioned above is the "longitude" which is the dihedral angle between (a) the Greenwich plane and (b) a plane perpendicular to the equator (containing the polar axis) and passing through the point described. Longitude is measured positive east of Greenwich. The other angle, the "latitude", is the angle formed by (a) a ray from the center of the earth through the point and (b) the projection of the ray on the equator plane. Latitude is measured positive north of the equator. There is, thus, a one-to-one correspondence between every point on the sphere and every ordered pair (θ, ϕ), where $0 < \theta \leq 2\pi$ and $\frac{-\pi}{2} < \phi \leq \frac{\pi}{2}$. ($\theta$ is called longitude and ϕ is called latitude).

If two points not collinear with the center of the earth are given, it is possible to define the "azimuth" of the path from the first to the second as follows: the azimuth is the dihedral angle between (a) the plane surface bounded by the ray from the center of the earth to the first point and the ray from the center of the earth to the north pole and (b) the plane surface bounded by the ray from the center of the earth to the first point and the ray from the center of the earth to the second point. The azimuth is positive if the second point is further east than the first and negative if the second is west of the first. If the first and second point have the same longitude or either (but not both) is at a pole, then the azimuth is zero if the second point is north of the first, and is π (not $-\pi$) if the second points is south of the first. Note that if the two points are collinear with the center of the earth, then the azimuth of the path from the first to the second is undefined.

A great circle is the intersection between the earth sphere and a plane through the center of the earth. The great circle distance between two points is the product of the radius of the earth (3440 n. mi.) and the angle (in radians) between rays joining the center of the earth and the two points. (NOTE: the angle is always less than or equal to π).

2.0 MATHEMATICS

If the longitude and latitude (θ and ϕ) of any point P are given, then a unit vector directed toward P can be expressed in "Cartesian" coordinates by the transformation equations:

$$\text{P:} \quad \begin{aligned} X &= \cos\phi \cos\theta \\ Y &= \cos\phi \sin\theta \\ Z &= \sin\phi \end{aligned}$$

A unit vector is one for which $X^2 + Y^2 + Z^2 = 1$. If two non-collinear unit vectors U_1 and U_2 are given, a vector P, normal to the plane containing them, is given by:

$$\text{P:} \quad \begin{aligned} X &= Y_1 Z_2 - Y_2 Z_1 \\ Y &= Z_1 X_2 - Z_2 X_1 \\ Z &= X_1 Y_2 - X_2 Y_1 \end{aligned}$$

This vector can be normalized (i.e., converted to a unit vector) by dividing each component by the vector length (the square root of the sum of the squares of its components). The unit vector, P', along P is given by:

$$\text{P':} \quad \begin{aligned} X' &= X/\sqrt{X^2 + Y^2 + Z^2} \\ Y' &= Y/\sqrt{X^2 + Y^2 + Z^2} \\ Z' &= Z/\sqrt{X^2 + Y^2 + Z^2} \end{aligned}$$

Now, P' is a unit vector normal to the plane containing U_1 and U_2. The direction of P' relative to the directions of U_1 and U_2 is the direction a right hand screw would advance if turned <u>from</u> U_1 <u>toward</u> U_2.

The angle between two vectors can be found by the following equation:

$$X_1 X_2 + Y_1 Y_2 + Z_1 Z_2 = \left(\sqrt{X_1^2 + Y_1^2 + Z_1^2}\right)\left(\sqrt{X_2^2 + Y_2^2 + Z_2^2}\right) \cos \zeta$$

where ζ is the required angle.

Note that if the vectors are perpendicular (i.e., $\zeta = \pi/2$) then $\cos \zeta = 0$ and

$$X_1 X_2 + Y_1 Y_2 + Z_1 Z_2 = 0$$

The dihedral angle between two planes is equal to the angle between two vectors normal to the planes, provided care is taken to be sure the directions of the normals are properly defined.

3.0 DESCRIPTION OF SUBROUTINE

Given the longitude and latitude of two points, write a subroutine named CALC to find:

I. The great circle distance between the two points (in nautical miles)

II. The azimuth of the path from the first to the second (in radians)

Further, given the longitude and latitude of a third point and an angle α, a small circle on the surface of the earth whose "radius" is α may be defined. Here α is not really the radius of the circle but is the angle between (a) the ray from the center of the earth to the center of the small circle and (b) the ray from the center of the earth to any point on the circle. Note that the great circle distance from the center of the small circle to any point on the circumference is the radius of the earth (3440 n. mi.) times the specified angle, α (in radians).

III. Find all the intersections (if any) of the great circle path connecting points 1 and 2 and the small circle defined by point 3 and list them in the order encountered as the path is traversed from point 1 toward point 2. Note that only points between 1 and 2 are desired.

The calling sequence for the subroutine should be:

CALL CALC (LOC1, LOC2, LOC3, ALFA, DIST, AZMUTH, NINT, INT1, INT2)

with

LOC1, LOC2, LOC3, INT1, and INT2 dimensioned 2,

where:

LOC1, LOC2, LOC3 are (LAT, LON) (in radians) of points 1, 2, and 3 respectively.

ALFA is the angle (in radians) defining the small circle.

DIST is the great circle distance (in nautical miles) from point 1 to point 2.

AZMUTH is the azimuth (in radians) of the path from point 1 to point 2.

NINT is the number of intersections of the great circle path and the small circle (0, 1 or 2).

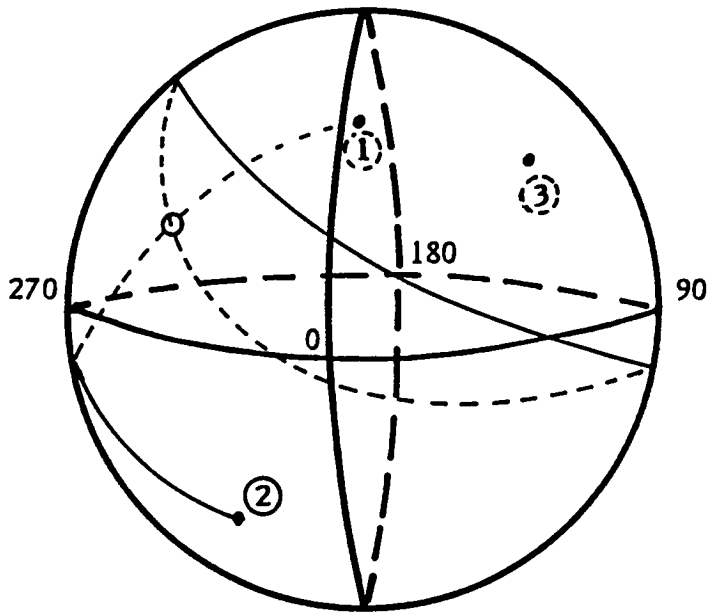
INT1, INT2 are (LAT, LON) (in radians) of the intersections. If NINT = 0, then INT1 and INT2 should be set to (0,0). If NINT =1 then INT2 should be (0,0).

4.0 SUPPLEMENTARY INFORMATION

The definition of the azimuth needs more expansion when the two points, P_1 and P_2, and the center of the earth all lie in one plane. In this case, the path from P_1 to P_2 will pass over a pole. (Remember that if P_1 and P_2 are collinear with the center of the earth, then the azimuth is undefined.)

In this case, first determine the shortest path from P_1 to P_2. Then determine the direction of this path when leaving P_1. If the direction is north, then azimuth = 0. Otherwise, the direction is south and azimuth = π.

APPENDIX G: PROBLEM #3 TEST CASE



	INPUT		OUTPUT	
①	LOC1:	0.692315	DIST:	9020.571289
		3.395404	AZIMUTH:	2.075972
②	LOC2:	-0.831659	NINT:	1
		5.836354	INT1:	0.197828
③	LOC3:	0.493327		4.068534
		2.400971	INT2:	0.0
	ALPHA:	1.561159		0.0

APPENDIX H: EXPERIMENT DATA FOR SUBJECT PROGRAM A1

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed seconds

1.	1 (1) 14.8	1 (2) 14.8	268 (3,5) 3035.9		
2.	1 (1) 14.8	37 (2) 289.7	116 (3) 1314.8	56 (5,6) 631.4	
3.	1 (1) 14.8	36 (2) 283.2	164 (5,6) 1857.2		
4.	1 (1) 14.7	19 (2) 152.3	249 (6) 2824.6		
5.	1 (1) 14.7	5 (2) 45.1	604 (4) 6851.0	217 (8) 2452.3	
6.	1 (1) 14.7	3 (3) 30.2	1 (2) 11.3	952 (4) 10,750.6	4092 (5,6) 46,164.0
7.	1 (1) 14.8	9 (2) 75.9	499 (4) 5664.8	244 (6) 2763.6	
8.	1 (1) 14.7	2 (2) 22.3	436 (4) 4943.4	68 (6) 769.0	
9.	1 (1,2) 14.8	491 (3) 5565.7	219 (5) 2472.9		
10.	1 (1) 14.8	17 (2) 137.2	278 (3) 3152.3	476 (4) 5381.4	342 (6) 3859.7
11.	1 (1) 14.7	18 (2) 144.5	282 (3,5) 3197.0		
12.	1 (1) 14.9	19 (2) 152.3	361 (3,7) 4092.5		
13.	1 (1) 14.9	13 (2) 106.7	291 (3) 3295.9	78 (4) 882.0	378 (10) 4269.0
14.	1 (1) 14.9	3 (2) 30.2	28 (3) 316.5	423 (4) 4775.4	2152 (8) 24,294.9
15.	1 (2) 14.6	1 (1) 14.8	36 (5,6) 406.8		

APPENDIX H: EXPERIMENT DATA FOR SUBJECT PROGRAM A1 (Continued)

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed seconds

16.	1 (1) 14.9	8 (2) 68.2	79 (3) 896.5	155 (4) 1752.2	533 (8) 6006.8	
17.	1 (1) 14.8	13 (2) 106.3	492 (3) 5574.4	26 (6) 292.3		
18.	1 (2) 14.7	1 (1) 14.9	113 (4) 1280.8	248 (5) 2812.6		
19.	1 (1) 14.7	13 (2) 106.7	1185 (5,8) 13,436.0			
20.	1 (1) 14.9	14 (2) 114.5	390 (4) 4418.5	970 (3) 10,997.0	888 (6) 10,018.9	
21.	1 (1) 14.8	1 (2) 14.5	121 (3) 1372.4	60 (6) 677.5		
22.	1 (1) 14.7	15 (2) 122.0	365 (4) 4136.5	1525 (3) 17,274.2	447 (5) 5039.5	1002 (9) 11,318.0
23.	1 (2) 14.8	1 (1) 14.8	933 (3) 10,574.3	331 (5) 3737.0		
24.	1 (1,2) 14.7	492 (8) 5580.9				
25.	1 (2) 15.0	2 (1) 22.3	446 (4) 5062.2	318 (8) 3598.4		
26.	1 (1) 14.7	5 (2) 45.6	338 (4) 3835.2	2126 (8) 24,091.9		
27.	1 (1) 14.7	8 (2) 68.2	346 (4) 3920.7	615 (8) 6974.6		
28.	1 (1) 14.9	11 (2) 91.6	1098 (4) 12,454.3	133 (5) 1503.7		
29.	1 (1) 14.6	5 (2) 45.0	95 (8) 1077.6			
30.	1 (1) 14.9	8 (2) 68.3	922 (6) 10,459.8			

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed seconds

31.	1 (1)	7 (2)	384 (4)	116 (5,6)		
	14.9	61.0	4353.3	1313.9		
32.	1 (1)	3 (2)	184 (5,6)			
	14.9	29.9	2085.7			
33.	1 (1)	6 (2)	195 (4)	633 (8)		
	14.7	53.6	2210.8	7180.8		
34.	1 (1)	19 (2)	38 (3)	397 (4)	2689 (5,6)	
	14.7	152.2	429.4	4484.5	30,366.7	
35.	1 (2)	1 (1)	70 (8)			
	14.7	14.8	792.1			
36.	1 (1)	19 (2)	1010 (6)			
	15.0	152.4	11,455.0			
37.	1 (1)	21 (2)	7 (4)	532 (3)	1205 (8)	
	14.6	167.9	78.4	6031.0	13,619.9	
38.	1 (1)	11 (2)	1311 (4)	337 (3)	367 (10)	
	14.9	91.4	14,862.0	3821.3	4148.5	
39.	1 (1)	1 (2)	31 (3)	717 (4)	1515 (10)	
	14.9	14.8	351.2	8100.0	17,132.9	
40.	2 (1)	6 (2)	251 (5,6)			
	22.3	52.9	2847.1			
41.	1 (1)	2 (2)	158 (4)	1326 (3)	258 (5)	278 (8)
	14.8	22.3	1790.3	15,024.1	2915.2	3138.1
42.	1 (1)	5 (2)	205 (3)	709 (4)	1307 (5,6)	
	14.8	45.0	2325.8	8011.6	14,753.5	
43.	1 (2)	1 (1)	841 (4)	9 (3)	539 (10)	
	14.5	14.8	9543.3	101.8	6082.6	
44.	1 (1)	4 (2)	80 (3)	203 (4)	868 (5,6)	
	14.8	37.7	907.2	2291.1	9797.8	
45.	1 (1)	6 (2)	375 (4)	1154 (3,5)	1200 (10)	
	14.9	53.1	4256.7	13,065.0	13,550.5	

APPENDIX H: EXPERIMENT DATA FOR SUBJECT PROGRAM A1 (Continued)

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed seconds

46.	1 (1)	21 (2)	521 (8)		
	14.7	167.8	5908.2		
47.	1 (1)	18 (2)	1155 (4)	385 (5,6)	
	14.9	144.5	13,088.0	4359.1	
48.	1 (2)	1 (1)	13 (3)	513 (4)	7 (5)
	14.8	14.6	147.3	5793.2	78.6
49.	1 (1)	24 (2)	45 (4)	1796 (3,5,6)	
	14.9	190.6	508.6	20,369.4	
50.	1 (2)	2 (1)	442 (4)	249 (6)	
	14.6	22.2	5013.7	2821.0	

APPENDIX I: EXPERIMENT DATA FOR SUBJECT PROGRAM B1

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed seconds

1.	4 (1)	1 (2,3)*	10 (4)	167 (8)		
	43.8	10.5	111.6	1872.9		
2.	3 (1)	1 (2,3)	46 (4)	142 (5)	298 (6)	
	32.7	10.6	514.8	1593.2	3325.9	
3.	7 (1)	1 (2)	3 (3)	27 (5)	310 (4)	3287 (7)
	76.9	10.5	32.9	302.1	3459.7	36,732.8
4.	2 (1)	2 (2,3)	12 (5)	166 (4)	3961 (6)	
	21.5	21.7	133.5	1851.9	44,254.6	
5.	4 (1)	2 (3)	1 (2)	7 (4)	337 (5)	581 (6)
	43.7	21.8	10.5	77.9	3784.8	6489.3
6.	6 (1)	5 (2)	1 (3)	14 (4)	776 (5)	5956 (8)
	65.9	55.0	10.5	156.2	8707.3	66,536.0
7.	12 (1)	5 (2)	2 (3)	114 (4)	793 (5)	150 (8)
	132.6	55.2	21.9	1279.8	8896.8	1673.0
8.	9 (1)	1 (2,3)	10 (4)	454 (5,6)		
	99.3	10.5	112.0	5093.0		
9.	3 (1)	3 (3)	1 (2)	108 (4)	185 (5)	660 (7)
	32.5	32.8	10.5	1210.8	2076.1	7366.8
10.	36 (1)	1 (2,3)	65 (4)	646 (5)	2607 (8)	
	399.4	10.4	729.6	7240.0	29,143.9	
11.	6 (1)	1 (2)	1 (3)	127 (5)	52 (4)	967 (6)
	66.0	10.4	10.6	1425.9	579.3	10,805.1
12.	2 (1)	3 (2,3)	52 (4)	119 (7)		
	21.5	32.9	584.1	1334.6		
13.	4 (1)	4 (3)	3 (2)	193 (4)	34 (5)	57 (6)
	43.8	44.3	32.8	2167.4	380.7	635.7
14.	8 (1)	3 (2)	1 (3)	74 (5)	7 (4)	1001 (6)
	88.2	32.8	10.5	830.0	77.3	11,190.1
15.	22 (1)	2 (2)	2 (3)	97 (4)	1152 (5)	4911 (8)
	243.8	21.6	21.7	1089.0	12,919.2	54,877.0

* For explanation regarding errors #2 and 3, see comments on pages 39 and 53.

APPENDIX I: EXPERIMENT DATA FOR SUBJECT PROGRAM B1 (Continued)

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed seconds

16.	2 (1) 21.6	5 (3) 55.5	5 (2) 55.3	39 (5) 437.6	13 (4) 144.2	814 (7) 9101.1
17.	1 (1) 10.5	6 (2) 66.5	10 (3) 111.3	66 (4) 740.8	381 (5) 4272.3	369 (6) 4125.7
18.	3 (1) 32.5	4 (3) 44.0	5 (2) 55.2	87 (4) 976.3	31 (5) 346.8	1963 (6) 21,943.1
19.	5 (1) 54.8	5 (2,3) 55.2	166 (7) 1861.9	193 (4) 2161.1		
20.	1 (1) 10.3	1 (2) 10.4	9 (3) 100.1	47 (4) 527.2	205 (5) 2297.4	215 (6) 2401.3
21.	12 (1) 132.6	2 (2,3) 21.7	67 (4) 752.1	492 (5) 5515.1	1461 (6) 16,343.9	
22.	11 (1) 121.4	2 (2,3) 21.7	321 (5) 3605.2	104 (4) 1160.7	3239 (7) 36,223.7	
23.	1 (1) 10.4	3 (2,3) 32.8	84 (4) 944.4	401 (5) 4497.4	3368 (8) 37,647.8	
24.	5 (1) 54.8	1 (2,3) 10.4	116 (4) 1301.9	677 (5) 7591.7	528 (8) 5903.9	
25.	5 (1) 54.9	1 (2,3) 10.5	2 (4) 21.8	25 (5) 279.6	2124 (6) 23,742.8	
26.	19 (1) 210.5	1 (2,3) 10.6	21 (4) 235.1	49 (5,6) 549.2		
27.	3 (1) 32.6	1 (2,3) 10.5	19 (4) 212.8	5.2 (5) 5744.3	1442 (6) 16,123.9	
28.	5 (1) 55.2	1 (3) 10.7	3 (2) 32.8	244 (4) 2739.3	323 (5) 3624.0	1683 (7) 18,809.7
29.	1 (1) 10.4	1 (3) 10.5	1 (2) 10.5	207 (4) 2324.8	108 (5) 1212.0	1919 (6,8) 21,448.0
30.	11 (1) 121.5	1 (2,3) 10.5	112 (4) 1257.6	48 (5) 537.9	449 (6) 5018.4	

APPENDIX I: EXPERIMENT DATA FOR SUBJECT PROGRAM B1 (Continued)

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed seconds

31.	2 (1)	2 (2)	1 (3)	24 (4)	50 (8)	
	21.4	21.6	10.7	269.0	559.8	
32.	6 (1)	1 (2,3)	11 (5)	62 (4)	755 (6)	
	65.9	10.6	123.0	691.4	8438.7	
33.	9 (1)	3 (3)	3 (2)	199 (4)	77 (5)	294 (8)
	99.4	33.0	32.6	2234.1	863.8	3283.0
34.	10 (1)	1 (2,3)	8 (4)	40 (5)	564 (8)	
	110.6	10.5	89.0	448.1	6302.9	
35.	11 (1)	1 (3)	4 (2)	165 (4)	77 (5)	3270 (6)
	121.6	10.5	44.3	1852.9	860.8	36,540.6
36.	2 (1)	3 (3)	2 (2)	156 (4)	768 (5)	1854 (7)
	21.5	32.9	21.6	1752.9	8613.8	20,728.5
37.	18 (1)	1 (2)	6 (3)	48 (4)	7 (5)	1313 (6)
	199.4	10.4	66.9	539.0	77.7	14,677.5
38.	2 (1)	1 (2,3)	8 (5)	117 (4)	373 (9)	
	21.4	10.4	89.3	1306.1	4168.9	
39.	2 (1)	1 (2,3)	93 (4)	518 (5)	1668 (6)	
	21.6	10.4	1044.2	5815.6	18,651.1	
40.	7 (1)	3 (3)	2 (2)	491 (4)	576 (5)	6030 (9)
	77.1	32.8	21.7	5519.2	6462.0	67,377.9
41.	8 (1)	2 (2)	3 (3)	154 (4)	104 (5)	415 (6)
	88.3	21.6	33.3	1730.0	1165.4	4632.5
42.	4 (1)	2 (2,3)	77 (5)	213 (4)	3387 (7)	
	43.7	21.7	866.0	2377.4	37,832.4	
43.	3 (1)	4 (2)	1 (3)	176 (4)	980 (5)	1560 (6)
	32.6	44.1	10.6	1976.9	10,990.4	17,406.5
44.	5 (1)	1 (3)	2 (2)	95 (4)	162 (5)	282 (8)
	54.8	10.7	21.7	1067.8	1816.5	3149.3
45.	1 (1)	2 (2)	1 (3)	50 (4)	267 (5)	1258 (6)
	10.4	21.6	10.4	562.1	2992.6	14,066.7

APPENDIX I: EXPERIMENT DATA FOR SUBJECT PROGRAM B1 (Continued)

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed seconds

46.	1 (1)	1 (2,3)	487 (4)	44 (5)	6372 (7)	
	10.4	10.5	5476.5	493.8	71,233.0	
47.	3 (1)	1 (2)	11 (3)	49 (4)	139 (5)	1642 (7)
	32.6	10.5	122.3	549.9	1557.9	18,357.2
48.	6 (1)	1 (2)	2 (3)	67 (4)	345 (5)	1582 (9)
	66.1	10.4	21.8	752.9	3868.5	17,689.7
49.	2 (1)	5 (3)	2 (2)	201 (4)	890 (5)	3702 (6)
	21.5	55.1	21.7	2260.3	9984.2	41,360.2
50.	13 (1)	1 (2)	1 (3)	371 (5)	17 (4)	1264 (6)
	143.8	10.5	10.5	4167.7	188.5	14,126.6

APPENDIX J: EXPERIMENT DATA FOR SUBJECT PROGRAM A2

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed seconds

1.	2 (1)	10 (2)	5 (3)	114 (4)	32,808 (5)
	.020	.148	.191	2.172	2952.103
2.	1 (1,2)	44 (4)	118 (3)	12,008 (5)	
	.012	2.121	14.750	267.359	
3.	1 (1)	5 (2)	1 (4)	180 (3)	7917 (5)
	.008	.080	.070	24.973	157.219
4.	1 (1)	5 (2)	17 (4)	41 (3)	216 (5)
	.008	.080	1.230	3.230	3.629
5.	1 (1,2)	20 (3)	103 (4)	5034 (5)	
	.008	.402	1.711	90.039	
6.	1 (1)	3 (3)	4 (2)	5 (4)	413 (5)
	.012	.049	.070	.090	13.320
7.	1 (1)	5 (2)	32 (4)	13 (3)	10,779 (5)
	.012	.070	.832	1.629	338.655
8.	1 (1)	1 (2)	25 (4)	21 (3)	3986 (5)
	.012	.021	.820	2.641	64.770
9.	2 (1)	14 (4)	10 (3)	6 (2)	7218 (5)
	.031	.209	.453	.160	267.996
10.	1 (1)	7 (2,4)	12 (3)	11,839 (5)	
	.012	.100	1.328	203.274	
11.	1 (1)	2 (2)	10 (4)	101 (3)	18,813 (5)
	.023	.031	.180	8.863	406.640
12.	1 (1)	4 (2)	41 (4)	21 (3)	8655 (5)
	.012	.100	2.051	2.840	143.086
13.	1 (1)	2 (2)	105 (4)	93 (3)	5,576 (5)
	.008	.039	2.621	11.359	132.016
14.	1 (1,2)	14 (4)	23 (3)	6645 (5)	
	.008	.301	3.539	145.703	
15.	1 (1)	2 (2)	26 (4)	92 (3)	2731 (5)
	.020	.051	1.422	5.449	41.523

APPENDIX J: EXPERIMENT DATA FOR SUBJECT PROGRAM A2 (Continued)

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed seconds

16.	1 (1) .012	4 (2) .121	47 (3) .980	49 (4) .879	3375 (5) 50.844
17.	1 (1) .008	11 (2) .250	27 (4) .590	27 (3) .887	2504 (5) 38.367
18.	2 (1) .031	6 (2) .150	7 (3) .129	50 (4) .898	3789 (5) 56.688
19.	1 (1) .012	6 (2) .150	47 (3) 2.410	132 (4) 2.293	245 (5) 3.680
20.	2 (1,2) .047	20 (3) .449	44 (4) .988	4121 (5) 62.219	
21.	1 (1,4) .012	10 (2) .320	127 (3) 6.770	10,894 (5) 249.656	
22.	1 (1) .008	1 (2) .020	19 (4) .328	6 (3) .090	2575 (5) 39.992
23.	1 (1) .008	12 (2) .270	7 (4) .281	25 (3) 1.660	
24.	1 (1) .008	3 (2) .070	22 (4) .461	38 (3) 3.910	
25.	1 (1) .012	2 (2) .039	50 (4) 1.031	16 (3) 1.520	
26.	2 (1) .031	5 (2) .141	14 (4) .223	2 (3) .262	
27.	1 (1) .012	2 (2) .061	10 (3) .332	26 (4) 1.137	
28.	1 (1) .020	1 (2) .051	64 (4) .961	30 (3) 1.301	
29.	1 (2) .020	1 (1) .010	20 (4) .289	52 (3) 1.113	
30.	1 (1) .020	7 (2) .180	66 (3) 1.000	1 (4) .027	

APPENDIX J: EXPERIMENT DATA FOR SUBJECT PROGRAM A2 (Continued)

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed seconds

31.	1 (1) .012	6 (2) .160	2 (4) .051	16 (3) .973
32.	1 (1,2) .020	31 (4) .531	117 (3) 9.379	
33.	1 (1) .020	1 (2) .039	9 (3) .438	106 (4) 1.891
34.	1 (1) .012	3 (2) .080	3 (4) .070	37 (3) 5.008
35.	1 (1,2) .008	1 (4) .020	1 (3) .023	
36.	1 (1) .012	7 (2) .158	2 (3) .031	5 (5) .090
37.	3 (2) .039	1 (1) .010	60 (4) 2.738	77 (3) 7.930
38.	2 (1) .039	3 (2) .080	26 (3) 1.520	1 (4) .023
39.	1 (1,2) .008	10 (4) .172	54 (3) 11.020	
40.	1 (1,2) .012	63 (3) 2.723	28 (4) .520	
41.	3 (1) .039	8 (2) .170	44 (3) 1.859	65 (4) 1.172
42.	1 (1) .020	2 (2) .029	42 (4) .637	39 (3) 9.219
43.	1 (1,2) .008	19 (4) .281	71 (3) 4.539	
44.	1 (1,2) .012	38 (4) .566	12 (3) 1.398	
45.	2 (1) .031	4 (2) .090	72 (5) 1.102	

APPENDIX J: EXPERIMENT DATA FOR SUBJECT PROGRAM A2 (Continued)

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed seconds

46.	1 (1) .012	1 (2) .020	20 (3) .410	15 (4) .277
47.	2 (1) .039	10 (2) .250	62 (3) 1.102	137 (4) 2.621
48.	1 (1) .012	3 (2) .080	1 (3) .031	29 (4) .527
49.	1 (1) .020	6 (2) .191	26 (4) .551	44 (3) 1.918
50.	1 (1) .020	2 (2) .070	8 (3) .129	145 (4) 2.500

APPENDIX K: EXPERIMENT DATA FOR SUBJECT PROGRAM B2

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed seconds

1.	2 (1,2)	1880 (3)	
	.148	85.055	
2.	7 (1)	14 (2)	6739 (3)
	.270	.230	247.984
3.	4 (1)	330 (2)	2262 (3)
	.078	6.090	98.914
4.	5 (1)	23 (2)	10,993 (3)
	.109	.387	288.757
5.	11 (1)	68 (2)	11,924 (3)
	.250	1.230	205.164
6.	8 (1)	48 (2)	5802 (3)
	.152	.980	104.773
7.	13 (1)	105 (2)	761 (3)
	.230	1.871	12.570
8.	2 (1)	9 (2)	19,628 (3)
	.031	.148	348.61
9.	2 (1)	10 (2)	5873 (3)
	.039	.188	99.078
10.	2 (1)	54 (2)	11,202 (3)
	.039	1.027	187.11
11.	2 (1)	41 (2)	537 (3)
	.051	.910	8.938
12.	1 (1)	116 (2)	1880 (3)
	.020	2.141	31.383
13.	6 (1)	15 (2)	4771 (3)
	.098	.223	82.906
14.	1 (1)	12 (2)	8638 (3)
	.008	.199	184.907
15.	4 (1)	23 (2)	116 (3)
	.063	.422	3.340

APPENDIX K: EXPERIMENT DATA FOR SUBJECT PROGRAM B2 (Continued)

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed seconds

16.	16 (1) .320	12 (2) .227	4067 (3) 93.25
17.	14 (2) .270	4 (1) .100	23,398 (3) 905.889
18.	17 (1) .363	34 (2) .621	11,416 (3) 238.797
19.	12 (1) .262	42 (2) .789	13,551 (3) 388.720
20.	19 (1) .363	33 (2) .621	14,895 (3) 835.157
21.	4 (1) .066	18 (2) .309	3074 (3) 86.406
22.	8 (1) .137	9 (2) .160	3853 (3) 164.168
23.	6 (2) .113	2 (1) .041	33,447 (3) 1418.960
24.	1 (2) .020	5 (1) .121	1532 (3) 25.430
25.	2 (1) .039	43 (2) .762	16,366 (3) 266.973
26.	10 (1) .188	2 (2) .031	1526 (3) 24.949
27.	2 (1) .031	10 (2) .180	37,447 (3) 621.170
28.	6 (1) .109	49 (2) .961	1532 (3) 25.69
29.	11 (1) .188	56 (2) 1.027	12,000 Truncated 234.92
30.	3 (1) .059	16 (2) .281	

APPENDIX K: EXPERIMENT DATA FOR SUBJECT PROGRAM B2 (Continued)

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed seconds

31.	8 (1) .211	28 (2) .551
32.	2 (1) .063	29 (2) .492
33.	18 (1) .391	14 (2) .219
34.	11 (2) .316	27 (1) .670
35.	3 (1) .051	37 (2) .699
36.	12 (1) .293	65 (2) 1.137
37.	17 (1) .480	40 (2) .730
38.	8 (1) .184	3 (2) .051
39.	4 (1) .141	38 (2) .648
40.	14 (1) .449	39 (2) .770
41.	3 (1) .090	56 (2) 1.039
42.	2 (1) .063	7 (2) .168
43.	7 (1) .148	2 (2) .039
44.	3 (1) .098	53 (2) 1.270
45.	13 (1) .352	119 (2) 2.090

APPENDIX K: EXPERIMENT DATA FOR SUBJECT PROGRAM B2 (Continued)

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed seconds

46.	6 (1) .160	6 (2) .098
47.	3 (1) .090	66 (2) 1.281
48.	5 (1) .109	56 (2) .988
49.	3 (1) .117	3 (2) .051
50.	2 (1) .031	12 (2) .211

APPENDIX L: EXPERIMENT DATA FOR SUBJECT PROGRAM A3

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed CRU's

1.	1 (1)	7 (2)	4 (6,10)		
	1	3	2		
2.	5 (1)	10 (2)	68 (6)	112 (5)	117 (3,4)
	3	5	32	63	61
3.	2 (1,4)				
	1				
4.	21 (1,2)	20 (6)	46 (5)	13 (3)	
	12	13	24	7	
5.	7 (1)	20 (5)	10 (3,4)		
	2	11	8		
6.	2 (1)	3 (3)	10 (5)	71 (6)	10 (7)
	2	2	6	34	5
7.	6 (1)	28 (6)	19 (5)	35 (7,10)	
	3	19	9	22	
8.	19 (1)	22 (5)	50 (6)	16 (8)	
	11	13	34	8	
9.	1 (1)	34 (2)	102 (7)	76 (5)	52 (6)
	0	16	49	54	34
10.	3 (1)	10 (2)	66 (6)	18 (7)	73 (10)
	1	6	32	8	34
11.	3 (1)	70 (2)	11 (5)	25 (4)	
	3	40	6	15	
12.	2 (2)	1 (1)	24 (6,10)		
	2	1	14		
13.	2 (1,5)	2 (2)	47 (8)		
	1	1	26		
14.	3 (1)	64 (2)	5 (5)	24 (6)	3 (4)
	4	36	3	11	2
15.	4 (1)	71 (7)	33 (6)	9 (10)	
	3	41	21	5	

APPENDIX L: EXPERIMENT DATA FOR SUBJECT PROGRAM A3 (Continued)

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed CRU's

16.	2 (2) 2	3 (1) 2	12 (5,7) 8	40 (6) 16	
17.	2 (1) 2	4 (2) 2	38 (4) 25		
18.	4 (1) 1	5 (2) 3	44 (6) 21	59 (5) 33	4 (10) 2
19.	12 (1) 7	12 (7) 8	18 (2) 12	26 (6) 16	13 (10) 9
20.	5 (1) 2	8 (6) 5	1 (2) 0	50 (5) 27	109 (8) 48
21.	1 (1) 1	13 (2) 7	20 (6,10) 8		
22.	3 (1) 0	8 (2) 3	9 (5) 5	52 (6) 29	42 (4) 19
23.	2 (1) 1	7 (6) 3	25 (2) 13	35 (4) 16	
24.	5 (1) 3	8 (5) 4	42 (2) 28	48 (6,10) 30	
25.	2 (1) 1	19 (6) 10	63 (5) 33	23 (2) 17	16 (3) 10
26.	1 (1) 1	1 (4) 0			
27.	3 (1) 2	2 (6) 1	2 (2) 0	12 (5) 4	53 (7) 32
28.	1 (1) 0	12 (5) 6	45 (7) 45	45 (2) 21	3 (4) 1
29.	1 (1) 0	4 (7) 1	34 (4) 22		
30.	2 (1) 1	60 (2) 28	34 (5) 13	153 (6) 84	38 (7,10) 24

APPENDIX L: EXPERIMENT DATA FOR SUBJECT PROGRAM A3 (Continued)

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed CRU's

31.	6 (1,2) 1	30 (7) 17	18 (5) 10	2 (6) 0	
32.	2 (1) 0	60 (2) 32	15 (5) 7	45 (6) 15	21 (3) 14
33.	13 (1) 12	46 (7) 22	21 (2) 12	100 (3) 58	37 (6) 18
34.	1 (1) 2	21 (6) 10	10 (7) 7	13 (5) 6	24 (2) 11
35.	2 (1) 2	5 (3) 3	2 (7) 0	15 (2) 7	13 (6) 9
36.	1 (1) 0	25 (2) 14	3 (6) 2	77 (7,10) 41	
37.	1 (1) 1	17 (5) 8	9 (2) 7	9 (6) 5	9 (10) 7
38.	2 (1) 2	2 (8) 1			
39.	5 (1) 3	1 (6) 1	51 (2) 26	58 (5) 26	209 (3) 102
40.	5 (1) 4	12 (5) 6	50 (6) 38	5 (2) 4	73 (10) 45
41.	3 (1) 1	40 (8) 21			
42.	1 (1) 0	8 (5) 6	1 (2) 1	23 (6) 13	17 (10) 9
43.	4 (1) 2	18 (6) 10	27 (5) 17	63 (2) 42	12 (8) 7
44.	10 (2) 8	3 (1) 1	7 (3,9) 5		
45.	1 (1) 1	11 (8) 5			

APPENDIX L: EXPERIMENT DATA FOR SUBJECT PROGRAM A3 (Continued)

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed CRU's

46.	3 (1)	70 (2)	151 (5)	34 (6)	125 (10)
	2	40	91	19	78
47.	5 (1)	12 (5)	57 (4)		
	2	5	45		
48.	6 (1)	20 (3)	7 (5)	23 (6)	85 (2)
	3	11	4	12	44
49.	2 (1)	19 (6)	9 (2)	4 (7,10)	
	1	11	5	1	
50.	1 (2)	3 (1)	16 (7)	15 (3)	13 (5)
	2	2	6	12	8

APPENDIX M: EXPERIMENT DATA FOR SUBJECT PROGRAM B3

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed CRU's

1.	3 (1)	21 (2)	145 (6)			
	2	14	62			
2.	2 (1)	23 (2)	157 (5)	388 (4)	75 (3)	206 (7)
	1	10	66	221	32	91
3.	4 (2)	2 (1)	54 (7)			
	3	2	21			
4.	4 (1)	4 (2)	9 (5)	77 (7)		
	3	2	5	43		
5.	1 (1)	23 (2)	111 (4)	28 (7)		
	1	15	52	12		
6.	1 (1)	23 (2)	162 (4)	34 (5)	755 (7)	
	1	10	74	18	354	
7.	3 (1)	5 (2)	51 (5)	259 (7)		
	2	0	30	139		
8.	2 (1)	16 (2)	31 (5)	312 (7)		
	1	9	13	166		
9.	3 (1)	2 (2)	7 (5)	80 (7)		
	2	1	2	53		
10.	1 (1)	23 (2)	190 (5)	83 (7)		
	0	14	97	44		
11.	2 (1)	23 (2)	60 (5)	166 (7)		
	0	15	26	94		
12.	4 (1)	10 (2)	60 (5)	554 (4)	318 (3)	1128 (6)
	2	5	34	312	154	520
13.	1 (1)	19 (2)	158 (5)	141 (4)	833 (7)	
	1	10	77	81	390	
14.	1 (1)	5 (2)	12 (5)	155 (4)	174 (7)	
	0	1	7	83	67	
15.	6 (1)	5 (2)	246 (5)	380 (4)	624 (7)	
	2	2	117	194	308	

APPENDIX M: EXPERIMENT DATA FOR SUBJECT PROGRAM B3 (Continued)

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed CRU's

16.	7 (1)	1 (2)	8 (7)			
	4	0	5			
17.	1 (1)	9 (2,4)	72 (7)			
	1	6	32			
18.	1 (1)	26 (2)	68 (4)	105 (5)	75 (7)	
	1	13	32	48	30	
19.	2 (2)	1 (1)	82 (7)			
	1	1	35			
20.	2 (2)	8 (1)	134 (7)			
	0	2	62			
21.	1 (2)	2 (1)	247 (5)	467 (7)		
	0	0	116	250		
22.	1 (1)	6 (2)	211 (5)	54 (4)	351 (7)	
	0	1	88	29	152	
23.	3 (1)	6 (2)	328 (7)			
	2	0	141			
24.	3 (2)	1 (1)	70 (6)			
	1	1	27			
25.	1 (1)	14 (2)	51 (5)	571 (7)		
	0	6	28	338		
26.	1 (1)	8 (2)	61 (4)	593 (5)	539 (3)	785 (7)
	1	5	26	238	225	364
27.	3 (1)	2 (2)	165 (7)			
	1	1	75			
28.	2 (1)	10 (2)	92 (5)	26 (4)	107 (7)	
	0	7	39	20	51	
29.	2 (1)	9 (2)	32 (7)			
	0	4	15			
30.	4 (1)	1 (2)	49 (5)	186 (4)	546 (7)	
	2	0	17	103	257	

APPENDIX M: EXPERIMENT DATA FOR SUBJECT PROGRAM B3 (Continued)

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed CRU's

31.	2 (1)	7 (5)	14 (2)	338 (7)		
	1	5	7	179		
32.	2 (1)	7 (2)	76 (5)	681 (7)		
	1	3	28	391		
33.	1 (1)	4 (2)	72 (5)	109 (7)		
	0	4	32	55		
34.	1 (1)	3 (2)	96 (7)			
	0	1	43			
35.	2 (1)	6 (2)	122 (4)	21 (5)	216 (3)	1101 (7)
	1	1	55	10	102	528
36.	7 (2)	1 (1)	69 (5)	231 (6)		
	2	1	32	117		
37.	2 (1)	5 (2,3)	114 (5)	236 (6)		
	1	2	52	90		
38.	2 (1)	1 (2)	459 (3)	14 (5)	405 (7)	
	2	1	197	7	185	
39.	4 (1)	6 (2,4)	134 (5)	1022 (7)		
	2	3	63	469		
40.	2 (1)	25 (2)	89 (6)			
	1	12	42			
41.	1 (1)	21 (2)	46 (7)			
	1	13	23			
42.	1 (1)	5 (2)	64 (5)	144 (7)		
	1	4	28	82		
43.	2 (2)	2 (1)	2 (4)	226 (6)		
	1	2	1	105		
44.	5 (1)	4 (2)	65 (7)			
	2	1	30			
45.	6 (1)	7 (2)	31 (5)	27 (6)		
	1	4	13	12		

APPENDIX M: EXPERIMENT DATA FOR SUBJECT PROGRAM B3 (Continued)

Explanation: Repetition no. Failure case no. (Error no.)
Time to failure in elapsed CRU's

46.	1 (1)	1 (2,4,5)	669 (7)		
	0	0	279		
47.	16 (1)	21 (2)	395 (5)	46 (7)	
	6	14	181	24	
48.	6 (1)	17 (2)	2 (5)	657 (4)	105 (6)
	3	8	1	360	50
49.	1 (1)	6 (2)	56 (7)		
	0	4	25		
50.	1 (1)	25 (2)	165 (4)	217 (7)	
	0	14	76	99	

1 Report No	2 Government Accession No	3 Recipient's Catalog No
NASA CR-165836		

4 Title and Subtitle	5 Report Date
Software Reliability: Repetitive Run Experimentation and Modelling	February 9, 1982
	6 Performing Organization Code

7 Author(s)	8 Performing Organization Report No
Phyllis M. Nagel James A. Skrivan	BCS-40366
	10 Work Unit No

9 Performing Organization Name and Address	11 Contract or Grant No
Boeing Computer Services Company Space and Military Applications Division P.O. Box 24346 Seattle, WA 98124	NAS1-16481
	13 Type of Report and Period Covered
	Contractor

12 Sponsoring Agency Name and Address	14 Sponsoring Agency Code
National Aeronautics and Space Administration Washington, D.C. 20546	

15 Supplementary Notes

Langley Technical Monitor: Gerard E. Migneault

16 Abstract

This paper reports on a software experiment conducted with repetitive run sampling. A run is a sequence of interfailure times recorded on each of a series of program states. Runs are replicated by reinitializing the program to the state of its original release and repeating the process of obtaining interfailure times on independently generated input data. This data has been used to verify that interfailure times are very nearly exponentially distributed and to obtain good estimates of the failure rates of individual errors and demonstrate how widely they vary. This fact invalidates many of the popular software reliability models now in use. It was observed that the log failure rate of interfailure time was nearly linear as a function of the number of errors corrected. A new model of software reliability is proposed that incorporates these observations.

17 Key Words (Suggested by Author(s))	18 Distribution Statement
Software reliability, software errors, software testing, reliability modelling, software experimentation	Unclassified - Unlimited

19 Security Classif (of this report)	20 Security Classif (of this page)	21 No of Pages	22 Price*
Unclassified	Unclassified	134	

* For sale by the National Technical Information Service, Springfield, Virginia 22161

End of Document